NPS-53-86-0001

NAVAL POSTGRADUATE SCHOOL

Monterey, California

LINEAR ALGEBRA USING PASCAL MT+

by

Larry Williamson

January 1986

Technical Report For Period

October 1982 - October 1984

Prepared for: Naval Postgraduate School
Monterey, CA 93943

NAVAL POSTGRADUATE SCHOOL
MONTEREY CALIFORNIA 93943

R. H. SHUMAKER
Rear Admiral, U. S. Navy
Superintendent

D. A. SCHRADY
Provost

This report was prepared by:

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NPS-53-86-0001	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Linear Algebra Using Pascal MT+	5. TYPE OF REPORT & PERIOD COVERED: Technical Report 10/82-10/84
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Larry Williamson	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE: January 1986
	13. NUMBER OF PAGES: 88
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

CPM86, IBM PC, linear algebra, simultaneous equations, eigenvalues, eigenvectors, roots of polynomials

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report includes the documentation and source listing of an user friendly interactive computer program that (1) solves simultaneous linear equations, (2) calculates the eigensystems of a square real matrix, and (3) finds the roots of a real polynomial. It is written in PASCAL MT+, the DRI implementation of Pascal, under the CPM86 operating system for the IBM PC. With the use of nested menus and keyboard filtering, the program requires no learning on the user's part. It channels the user in the right direction. Input matrices are entered from the keyboard while the answers maybe outputted to

20. ABSTRACT (Continuation)

the screen and/or printer. The documentation along with the source listing describe how to develop a large, user friendly program through the use of modular techniques such as overlays.

LINEAR ALGEBRA USING PASCAL MT+

Introduction to Linear Algebra Calculator

This program solves the two major problems of linear algebra: (1) the determination of the eigenvalues and eigenvectors of a square matrix "A" and (2) the solution of simultaneous equations "A X = B" for an n by n invertible matrix "A" and any n by m matrix "B". One must use the built-in matrix editor to enter the matrices "A" and "B". A program , based on eigenvalues of a companion matrix, to calculate the roots of a polynomial with real coefficients is also included. The source code of the mathematical routines and I/O routines could be used to build a more elaborate matrix calculator that might include disk I/O, a more flexible editor, addition and multiplication of matrices, and a matrix expression evaluator. This program does not have these capabilities.

The hardware configuration necessary to run this program is an IBM PC with 128K of RAM memory and one double-sided drive. It is written in Pascal MT+ and thus the source code could be compiled with any MT+ compiler after making a few modifications to the system I/O routines. Depending upon what floating point library is linked in with the compiled code, the program may be run with an 8087 chip or an 8088 chip.

To run the program one may do any one of the following: (1) if the machine is off, insert the progam diskette in drive A and

turn the power on; CPM-86 will be loaded and the program, called MATRIX, will be executed; (2) if the machine is on, insert the diskette in drive A and do a warm boot by pressing "Ctrl-Alt-Del" simultaneously; CPM-86 will be loaded and the program will be executed; or (3) if CPM-86 has been loaded, insert the diskette in drive A, make the CPM prompt say A>, and key in MATRIX followed by carriage return. There are obviously other possibilities. The program and all its overlays could be transferred to RAM disk M and executed from there with M also the default drive. One must always have the program and its overlays on the default drive because the program looks for its overlays on the default drive.

Brief instructions on how to use the program will now be given. When the program is executed, the main menu is displayed. By pressing "G", one can view some very brief instructions. They are intended for somebody who has never run the program before or who does not like to read documentation. Before doing any matrix work, one must press "F" to bring up the matrix editor and displayer. The menus are self-explanatory and channel the user in the desired directions. To do matrix work one must enter a matrix "A" but the entering of "B" is only necessary for solving "AX = B". Only one matrix "A" and one matrix "B" may be stored at a time. To enter a new matrix after one has already been created one must answer "Y" to the erase prompt.

When matrices "A" and "B" are first created, they are initialized at all zeros. The user then edits the matrix in increasing column order. Each column will be displayed for

editing. If one answers "Y" to the column correct prompt, then the next column will be displayed for editing. If one answers "N", the cursor will be positioned at the entry in the first row of that column. If one wants to change that entry, he enters a new number followed by return. If one simply enters return, the entry is not changed and the cursor is positioned at the next row. After the entire column has been edited, the column is redisplayed with the same prompt. To edit column 4, one is forced to first view and answer "Y" to the first three column correct prompts. A compromise on flexibility and editing speed for user friendliness and less programming has been made here.

Once "A" has been entered, one may return to the outer menu to calculate the inverse, determinant, eigenvalues, and eigenvectors. If "B" has been entered also, one can press "C" to solve a linear system "A X = B". One can solve for the roots of a polynomial by pressing "A". One enters the coefficients of the polynomial within this segment of the program, not in the editor.

Two possible I/O problems are the following: (1) real numbers are not filtered for underflow or overflow when inputted; they have to be entered in the interval [1.0 E-307, 1.0 E+307]; and (2) the printer must be on-line (or must be put on-line when the CPM-86 error message is displayed) if a hardcopy is desired. If one cannot get the printer on line after a hardcopy was requested, CPM-86 exits the user back to a system prompt and the entered matrices "A" and "B" are lost.

An example matrix "A" to key in is : column 1 is the vector (1,3,6), column 2 is the vector (-3,-5,-6), and column 3 is the vector (3,3,4). Let X and r (not variables in the Program Matrix) stand for the associated eigenvector and eigenvalue in the equation "AX=rX". The eigenvalues of "A" are 4,-2, -2 and the associated eigenvalues are respectively (1,1,2), (-1,1,2) and (1,3,2). It should be noted that eigenvectors returned by the calculator will differ by scalar multiples from those above. Also, keep in mind instead of (-1,1,2) and (1,3,2) which are associated with the eigenvalue -2, we could have another pair of eigenvectors such as (0,4,4) and (2,2,0) which are in the same eigenspace as the first pair. The inverse of the matrix "A" is the matrix whose columns are: (1) .125(-1,3,6), (2) .125(-3,-7,-6), and (3) .125(3,3,2). This matrix has 16 as its determinant. The polynomial X**3-6X**2+11X-6 would be entered as the vector (1,-6,11,-6). It has roots 1,2,3.

Description of CRT and Keyboard utilities

The utility programs in Module CRTLIB are for cursor control and filtered input of characters, integers, and real numbers. This section should be read concurrently with the source listing of Module CRTLIB. These routines are based upon two CPM-86 BIOS calls, which are implemented in the Procedure Bioscall. Bioscall has two input parameters: (1) FUNC, a variable of type CPMOPERATION and (2) OCH, an integer variable. Bioscall, as presented here, only responds to two inputs, CONIN and CONOUT, of

type CPMOPERATION.

In the following, terminal character refers to a character that is being sent to the monitor to effect a cursor control. To send a terminal character to the CRT interface(monitor), one calls Bioscall with input parameters FUNC equal to CONOUT and OCH equal to the ASCII number of the terminal character. The value of the variable BDOSVAL(integer global to Module CRTLIB) is not used explicitly; here it is used only to call Procedure @BDOS86. To read a character from the keyboard, one calls Bioscall with input parameters FUNC equal to CONIN and OCH equal to any dummy integer. The global variable BDOSVAL is set equal to the ASCII number of the keyboard character pressed. The number 50 in @BDOS86(50, ADDR(DESCRIPT[1])) is the BDOS function number for a BIOS call. The five consecutive bytes allocated by DESCRIPT are used for passing information in the @BDOS86 call. For example, "DESCRIPT[1]:=4" is the Pascal line for console display while "DESCRIPT[1]:=3" is the line for keyboard input. The Pascal MT+ manual gives a brief description of the MT+ utility @BDOS86 while the CPM-86 manual gives a detailed discussion of BDOS calls in its Appendix D. The Procedure Bioscall presented here is a Pascal implementation combining @BDOS86 with the BDOS calls(for function number 50) of CPM-86.

One could expand Procedure Bioscall to include all the parameters in CPMOPERATION by combining the ideas in Bioscall with Appendix D of the CPM-86 manual. One does not need to use Bioscall to write an alpha-numeric character to the screen; Write or Writeln can be used for this. Most of the routines in Module

CRTLIB either call Bioscall or another routine which calls Bioscall. Function Getchar is the only procedure that calls Bioscall with the CONIN parameter.

In order to use the utilities in Module CRTLIB one must call Procedure Crtinit at the beginning of the main program. Crtinit initializes the arrays CRTINFO and PREFIXED so that their values can be used by the utilities in Module CRTLIB.

Function Getchar, using Bioscall, performs the task of reading a character from the keyboard. The input to Getchar is a variable set of characters, called OKSET in Getchar's declaration. When Bioscall is called by Getchar to get a character from the keyboard, Getchar then checks to see if the character represented by BDOSVAL is in OKSET. If it is, this character becomes the value of the Function Getchar and is used in the procedure calling Getchar; if it is not, a beep is sounded and the process is repeated until a character in OKSET is finally entered. This version of Getchar automatically changes lower case to upper case. An example of Getchar being used with OKSETs that are dynamically changing can be seen in Procedure Getreal.

Procedure Crt is the main routine that calls Procedure Bioscall with the CONOUT parameter. Depending on the boolean value given by an element in the array PREFIXED, the leadin character, ESC, is sent to the monitor and then the actual terminal character is sent to the monitor. For example, suppose we want to clear the screen. One uses the variable ERASEOS of type CRTCOMMAND as the parameter input to Procedure Crt. Using

ERASEOS as the index variable, Crt sends the leadin character ESC to the monitor since PREFIXED[ERASEOS] is true and then sends the character 'J', which is CRTINFO[ERASEOS]. It should be noted that in the above, the ASCII numbers of the various characters are actually inputted to Bioscall in Procedure Crt.

One can control background colors by using the type COLOR, the arrays COLORINFO, CRTINFO, and PREFIXED along with the Procedures Initcolor, Altcolor, and Paint. One then needs to write a procedure similar to Altcolor to change foreground colors. These would be the fundamental library routines for using colors.

Procedure Gotoxy(X,Y:INTEGER), which also calls Bioscall with input parameter CONOUT, places the cursor at vertical line number X and horizontal line number Y. The other cursor control routines are fairly obvious in view of the above discussion.

Procedure Intread is used to read an integer between -32768 and +32767. The characters are filtered, put into a string, checked for the proper range, and then converted to an integer. Intread requires an integer to be entered; one cannot simply enter carriage return. Procedures Getreal and Value are used together to read a real number. Some explanation is required on the use of Getreal. In the procedure calling Getreal, we have a string variable, SREEL, initialized at '', an empty string of length zero but still a string. Procedure Getreal reads and dynamically filters a string of characters and if this string is of length greater than zero, this string is returned to the

calling procedure by Getreal and SREEL is set equal to it. If Getreal only reads a carriage return, then SREEL remains only an empty string and Procedure Value is not called. If SREEL has length greater than zero, Procedure Value calculates the real number corresponding to SREEL. Getreal and Value are used to edit a real number that already exists, either through initialization or calculation. By pressing carriage return, one can leave an existing real number as is. This use eliminates a computer crash caused by keying in only a carriage return when the usual READ or READLN wants an actual number. Procedure Value was taken from The Byte Book of Pascal while Procedure Powrten is from the Pascal Users Manual by Jensen and Wirth.

Data and Overlay Description

This program illustrates the use of module overlays, an ordinary module, matrix types and external variables. This description explains the manner in which the program was implemented, and therefore it should be used as a guide to understanding the structure of the code. Modules are separately compiled groups of procedures. There are two types: (1) an overlay module and (2) an ordinary module which must be linked with an overlay module or main program. The Module CRTLIB is separately compiled and linked in with the main program. CRTLIB's subroutines essentially become a part of the main program(sometimes called the root overlay) and always are resident in RAM memory. Since the utilities in CRTLIB are continually being

called, it would not be practical to make CRTLIB an overlay module and thus force a lot of disk accesses. If a module overlay(including the main program) uses a procedure in Module CRTLIB, then the "calling" overlay must declare the procedure as an external procedure in the calling overlay's declaration heading just below the variable declarations.

The data storage of the program will now be explained. The controlling data allocation declarations are (1) the TYPE DOMAIN1=1..20, (2) the CONST MAXDEG=20, (3) the CONST MAXDEGP1=21, and (4) the TYPE DOMRP1=1..MAXDEGP1. MAXDEG, MAXDEGP1(=MAXDEG PLUS 1), and DOMRP1 are used to solve for roots of a polynomial. DOMAIN1 is the maximum dimension of a matrix entered as input data. MAXDEG should be set to be less than or equal to the value of the DOMAIN1 upper bound since the polynomial root finding technique uses an n by n matrix to solve an n degree polynomial equation. In this program, the coefficients of a polynomial are not stored in a global variable(for the main program) and there is not much I/O to go with the polynomial routine. It is included as a side benefit to illustrate the use of the eigenvalue procedures as subroutines to another procedure. Variables that are referred to as global are declared in the main program and then declared external in the overlays that use them. The matrix "A" whose square dimension is IONDIM is stored in the upper left-hand corner of the global variable IOTA and the matrix "B" whose dimension is ROWDIMB by IOM is stored inthe upper left-hand corner of the global variable IOTB. IOTA, IOTB, IONDIM, ROWDIMB, and IOM are declared in the

root overlay(Program Matrix) and referenced in the appropriate overlays by declaring them as external variables. The global matrix "XX" is used to store the solution to "AX=B" and is referenced as external in Module Overlay20(whose source code is AXEQSB.SRC). The global vectors TEVR, TEVI store the real and imaginary components of the eigenvalues and are referenced as external in Module Overlay24(EIGENVEC.SRC). The global matrix TVEC, referenced as external in Module Overlay24, is used to store the eigenvectors of the matrix "A". Communication of data between the root overlay (Program Matrix) and the overlay modules #3,#19,#20,#24 is accomplished by initially declaring the above mentioned variables in the main program and by declaring them external in the necessary overlay modules. The Types of matrices and vectors must be declared in the root overlay and redeclared in the overlay modules. When using dimensions less than the maximum allocated in the Types, all data will be stored in the upper left-hand corner of matrices or beginning components of a vector. Intermediate results produced by Module Overlay1 (LINSYS.SRC), Module Overlay5 (EIGENHQR.SRC), and Module Overlay6 (EIGENBAL.SRC) are passed as parameters in the procedure calls.

The root overlay (Program Matrix) calls directly six main procedures located in six overlays. They are (1) Procedure Matrixio, located in Module Overlay3, which is the editor for matrix input, (2) Procedure Help, located in Module Overlay23, which is a brief set of instructions on how to run the program, (3) Procedure Ttrdet, located in Module Overlay19, which calculates the determinant of the matrix stored in variable IOTA,

(4) Procedure Ttsaxb, located in Module Overlay20, which solves "AX=B" for "A" stored in the variable IOTA and "B" stored in the variable IOTB or "B" is the identity matrix(used to calculate the inverse of "A"), (5) Procedure Ttrnaa, located in Module Overlay24, which calculates eigenvectors and eigenvalues of the matrix stored in IOTA, and (6) Procedure Ttrpqr, located in Module Overlay21, which calculates the roots of a polynomial with real coefficients. These six procedures which are located in module overlays must be declared external in the root overlay heading. After the word "External", the overlay number to which the procedure belongs must appear in square brackets.

Other external procedures used within these overlays do not have to be declared within the root overlay; they are declared as external procedures with the appropriate overlay numbers within the overlay that calls them. For example, Procedure Ttrdet, located in Overlay Module19, calls Procedure Rlud, located in Overlay Module1. Therefore, Procedure Rlud must be declared external in the heading of Module Overlay19. Module Overlay3 uses procedures located in the Module CRTLIB(not an overlay module), and therefore these called procedures must be declared as external in the heading of Module Overlay3. These external procedures from Module CRTLIB do not have overlay numbers.

Module Overlay3 (IOMATRIX.SRC) is the module responsible for keyboard input of matrices and printer output of matrices. The maximum size of the matrix is set in the TYPE DOMAIN1 = 1..20. If one wanted only matrices of dimension 10 or less, one would set TYPE DOMAIN1 = 1..10. One must change these type declarations for

all the overlays that use them. It should be noted that the editing programs are based upon editing a column of a matrix that fits onto one screen display. If one wanted to enter a column of dimension larger than twenty, the column display and editing routines would have to be modified a little. On the other hand it is not practical to enter a matrix larger than 10 by 10, let alone 20 by 20. Large matrices should be generated by a program or read from a disk file. For example, Module Polyroot generates a 20 by 20 matrix when one requests the root of X**20=1.

Within this module, Procedure Getcolumn and Procedure Getpolcof are the work horses. Respectively, they take the column of a matrix or the coefficients of a polynomial and display them on the screen for editing. These two routines use the cursor control utilities and Procedures Intread, Getreal, and Value to edit existing columns. Procedure Matrixio is the editing procedure which calls the Procedures Edit and Display. Edit displays menus, initializes matrices, and calls the above mentioned procedures to edit existing matrices. Procedure Display, which is somewhat redundant, gives a faster display than Procedure Edit. Procedure Hardcopy is used to printout the matrices "A", "B", inverses, determinants, matrix equation solutions, eigenvectors, and eigenvalues.

Module Overlay24 (EIGENVEC.SRC), Module Overlay6 (EIGENBAL.SRC), and Module Overlay5 (EIGENHQR.SRC) are the overlays which calculate eigenvectors and their corresponding eigenvalues. Procedure Ttrnaa takes the global matrix IOTA

containing the matrix "A" and its dimension, IONDIM, and calls the Procedure Rnaa. Procedure Rnaa is the procedure which calls the Procedures Balance, Elmhes, Eltran, Hqr2, and Balbak, which together calculate the eigenvalues and eigenvectors. The eigenvalues are returned to Ttrnaa in the global arrays TEVR and TEVI. TEVR[I] and TEVI[I] give the real and imaginary parts of the eigenvalue #I. The eigenvectors are returned in the matrix TVEC in a not so obvious way. They are returned in an order corresponding to the eigenvalues in TEVR and TEVI. The manner in which TVEC returns the eigenvectors needs to be illustrated by an example. Let TEVR[1]=1 and TEVI[1]=2; the eigenvalue is 1+2i. Then TEVR[2]=1 and TEVI[2]=-2 since eigenvalues and eigenvectors of real matrices occur in conjugate pairs. Now the two eigenvectors that correspond to these eigenvalues are conjugates. TVEC in its first column will have the real part of eigenvector #1 and in its second column it will have the imaginary part of eigenvector #1. Eigenvector #2 will just be the conjugate of this eigenvector. If the eigenvalue is pure real, then the eigenvector is pure real and only one column is necessary. In the case of a matrix with repeated eigenvalues without a full set of eigenvectors, the eigenvectors will be repeated in the matrix TVEC(probably numerically slightly different). These eigenvector subroutines are Pascal translations by the author of EISPACK eigenvector routines in Fortran.

Module Overlay21 (POLYROOT.SRC) also uses the eigenvalue subroutines to solve for the roots of a polynomial with real coefficients. It makes the integer 1 the leading coefficient of a

polynomial of degree n and enters the other n coefficients into the first row of an n by n matrix with 1's on the subdiagonal; this matrix is called the companion matrix and the eigenvalues of it are the roots of the polynomial. Procedure Ttrpqr does the preliminaries and Procedure Rpqr calls the sequence of eigenvalue subroutines to find the eigenvalues which are returned in the variables TWR and TWI. Procedure Balbak is not used since the eigenvectors are not needed. POLYROOT.SRC illustrates the use of these subroutines in a real problem and it also serves as a test of the correctness of the eigenvalue codes.

Module Overlay20 (AXEQSB.SRC) and Module Overlay1 (LINSYS.SRC) are the overlays which solve systems "AX=B" as well as find the inverse of the matrix "A" (B will be the identity matrix in this case). Again the "A" matrix is in IOTA and the "B" matrix is in IOTB. Procedure RLUD does an L-U decomposition on the matrix in the variable SA(the constant matrix SB tags along) and Procedure Rfbs does the back substitution to solve the system. In the case where "B" has more than one column, Rfbs is called IOM times to solve the equations. Procedures Rlud and Rfbs are Pascal translations by the author of a Sandia Labs matrix equation solver; it is very successful with matrices "A" that are almost singular(no inverse).

Module Overlay19 (DETERM.SRC) and Module Overlay1 (LINSYS.SRC) are used to calculate determinants. Rlud in LINSYS.SRC does an L-U decomposition of the matrix TALU(a copy of IOTA) and returns the L-U decomposition in TALU. The matrix L has ones on the diagonal and therefore the determinant of "A"

will be the determinant of "U" which is just the product of U's diagonal entries.

Layout of Overlays

Module overlays are divided into groups. Overlays #1 -#16 are in overlay group #1, overlays #17-#32 are in overlay group #2, etc.. The source code of an overlay module must begin with its identification number; for example overlay module #1 has the heading, Module Overlay1. The root overlay is just the main program; here it is Program Matrix. This program does not use the most sophisticated overlay structure. In this program, overlays within the same group do not call each other and there is no heap. When compiling an overlay, one does not have to specify the address in RAM memory where the overlay will be loaded. This is done during the linking process. All the overlays within the same group must be located at the same address. This address must be larger than the length of the root overlay including any runtime library routines or ordinary modules linked with the root overlay. For example, the root overlay, Program Matrix with routines from the libraries TRANCEND.R86, 87REALS.R86, PASLIB.R86, and Module CRTLIB use 6FFCH(hex) bytes of storage. The memory allocated for overlay group #1 must be some address larger than this; here 8600H is used. In the linking instructions below, overlays #1 - #16 must be directed to begin at 8600H. Then one looks at the largest length of the compiled overlays in group #1 and adds this to 8600H. In this program, Module Overlay5 has length 42AAH. This should be verified again after all the

overlays in group #1 have been linked to the root overlay. One adds 8600H to 42AAH to obtain the length C8AAH. The address where overlays #17 - #32 begin must be larger than C8AAH. This program uses address D200H for the location of overlays in group #2. The overlay of the largest length in group #2 is Module Overlay21 with a length of 1370H. 1370H plus D200H equals E570H. Therefore, the total length of the code that must be specified in the last link command with the R switch must be greater than E570H. The value used is FC00H. The data storage required is only that from those variables declared in the main program and it is 576CH. The value 8000H is more than adequate and is declared with the D switch in the last linker command. Local variables and parameters are stored in the stack. This stack is automatically given 32k bytes of RAM. One can actually assign the stack size by using the Z switch in the last linker command. For example, "Z/300" will assign 3000H for stack memory. Note that 300 here means 3000H or 12K bytes.

One has to make sure the total RAM allocated for data, code, stack, and heap is smaller than the RAM allocated for program execution. Otherwise, upon program execution, a "Memory not available" message will be displayed. In this program, the stack was reduced to 12K in order that the program would execute in a 128K machine. Having a stack too small is a potential problem, but by far the major source of problems is having one overlay group overwrite another. Apparently, one needs to have some memory available between the end of the largest overlay in a particular group (including the root overlay) and the beginning

of the next overlay group. For example, in this program when the addresses 7200H, B800H, and DE00H were used in place of the addresses 8600H, D200H, and FC00H the program would not execute properly. The Pascal MT+ manual should address this problem. If one experiences weird program execution, the distance between overlay groups should be increased as the first approach to solve the problem.

Computer Configuration for Pascal MT+

In doing developmental work for this program, the IBM PC was configured with two DS/DD disk drives, 512K of RAM, and the CPM-86 operating system. The RAM was partitioned into 140K for running programs(compiler, editor, linker, and the executable linear algebra routine) and 372K for a RAM disk, called the M: drive. The program SETUP.CMD, furnished with CPM-86, was used to configure the RAM drive and the cursor pad of direction arrows. Upon bootup, the RAM drive is created and the cursor pad is initialized.

A RAM disk is essential (though not required) for program development with Pascal MT+ because of the heavy disk access by the MT+ Speed Editor, Compiler, and Linker of their overlays and the program's source and object code files. If one follows the implementation techniques described in the following, the user will find MT+ not just a powerful Pascal implementation, but one that possess good development time and is not cumbersome. It should be mentioned that in many articles comparing the MT+

implementation to others, the benchmarks involving compile and link times are misleading. Pascal MT+ uses a lot of disk I/O because of its code size and this eats up a lot of time. Using a RAM disk reduces this I/O time considerably.

One must configure three diskettes. Diskette #1, called the compiler diskette, must contain CPM.SYS(configured for RAM drive and optionally the cursor pad), the MT+ compiler(consisting of a CMD file and its several overlays), and three R86 files: (1) 87REALS.R86, (2) PASLIB.R86,and (3) TRANCEND.R86. It is optional, but useful to put the utility routines such as PIP.CMD and STAT.CMD on the compiler diskette. Diskette #2, called the editor diskette, should contain the Speed Programmming Editor(CMD file and its overlays), SIP.CMD(PIP.CMD renamed by the user), and STAT.CMD. Diskette #3, called the linker diskette, should contain the MT+ linker(CMD file and its overlays) and the SUBMIT.CMD file. For single stroke execution, it is convenient to rename the compiler, editor, and linker respectively M.CMD, S.CMD, and L.CMD.

The following is the start up process. Put the compiler diskette in drive A: and the editor diskette in drive B: and then turn the power on. When the "A>" prompt comes up, key in "PIP M:=B:S*.*". This will transfer the editor, SIP.CMD, and STAT.CMD to the M: drive. One can use an editor other than DRI's Speed Editor. Users of UCSD's P-Editor will find the Speed Editor acceptable when not in the P-system. After the editor is transferred to RAM drive, one removes the editor diskette from drive B: and inserts a work diskette containing source and their

compiled, R86, files.

One then presses Control-C and changes the default drive to M:. If one wants to edit an existing source file, one uses SIP.CMD on drive M: to "pip" the source file from drive B: to drive M:. While in the M: drive, one edits a Pascal source file. If one is using the Speed Editor, a word of caution is necessary. If one attempts to save a file on a drive without enough memory(including RAM drive), the old source file as well as the source file in the editor's buffer will be lost. This potentially serious problem can be avoided if one is aware of it. If the Speed Editor is used one should use its syntax scanner and variable checker to locate syntax errors and typos prior to compilation.

Once a source file has been edited it must be compiled. It is not necessary to transfer the compiler to RAM drive. One must change the default drive from M: to A:, leaving the compiler diskette in drive A: and the source file on RAM drive. Since Pascal MT+ is oriented toward modularity and modular compilation in particular, individual source files(modules and overlay modules) are relatively short. Because of this, disk I/O to drive A: to load the compiler overlays is not too frequent. With the source code being read from RAM drive and compiled code being written to RAM drive, compiles are quite timely.

To compile all the overlays and modules for the linear algebra calculator, one must do the following tasks. There are eleven source codes which must be saved on the work disk. They

are (1) MATRIX.SRC, (2) CRTLIB.SRC, (3) IOMATRIX.SRC, (4) EIGENBAL.SRC, (5) EIGENHQR.SRC, (6) MHELP.SRC, (7) LINSYS.SRC, (8) DETERM.SRC, (9) AXEQSB.SRC, (10) POLYROOT.SRC, AND (11) EIGENVEC.SRC. Each file should be transferred to RAM drive and then compiled. To compile the root overlay, MATRIX.SRC, one enters the following command: "MT+86 M:MATRIX.SRC". If one changed the name of the compiler to M.CMD, then one enters "M M:MATRIX.SRC". The compiler will compile the file and write a code file, MATRIX.R86, to RAM drive. Similarly, one compiles the other ten source code files to produce ten more R86 code files. These code files should be saved on the work diskette in drive B:. It should be emphasized that if the source code of a module or overlay module is not changed, then it does not have to be compiled.

The most important implementation method in MT+ code development is to automate the linking process. Without this automation, the linking process is extremely time consuming. Automatic linking is accomplished by using SUBMIT.CMD and linker input command files, called KMD files. These will now be described in detail and applied to the linear algebra calculator. Once all the overlays have been compiled one makes up a KMD file for each compiled overlay module. A KMD file is actually a text file keyed in using an editor and given the suffix KMD instead of the usual SRC suffix. The following KMD text files are needed:

(1) M0.KMD with the line of source:

M:MATRIX,M:CRTLIB,M:TRANCEND,M:87REALS,M:PASLIB/S/E/W

(2) M1.KMD with the line of source:

```
M:MATRIX.001=M:MATRIX/O:1,M:LINSYS,M:PASLIB/S/P:8600/L
```

(3) M3.KMD with the line of source:

```
M:MATRIX.003=M:MATRIX/O:3,M:IOMATRIX,M:PASLIB/S/P:8600/L
```

(4) M5.KMD with the line of source:

```
M:MATRIX.005=M:MATRIX/O:5,M:EIGENHQR,M:PASLIB/S/P:8600/L
```

(5) M6.KMD with the line of source:

```
M:MATRIX.006=M:MATRIX/O:6,M:EIGENBAL,M:PASLIB/S/P:8600/L
```

(6) M23.KMD with the line of source:

```
M:MATRIX.017=M:MATRIX/O:23,M:MHELP,M:PASLIB/S/P:D200/L
```

(7) M19.KMD with the line of source:

```
M:MATRIX.013=M:MATRIX/O:19,M:DETERM,M:PASLIB/S/P:D200/L
```

(8) M20.KMD with the line of source:

```
M:MATRIX.014=M:MATRIX/O:20,M:AXEQSB,M:PASLIB/S/P:D200/L
```

(9) M21.KMD with the line of source:

```
M:MATRIX.015=M:MATRIX/O:21,M:POLYROOT,M:PASLIB/S/P:D200/L
```

(10) M24.KMD with the line of source:

```
M:MATRIX.018=M:MATRIX/O:24,M:EIGENVEC,M:PASLIB/S/P:D200/L
```

(11) MR.KMD with two lines of source:

```
M:MATRIX,M:CRTLIB,M:TRANCEND,M:87REALS,/C

M:PASLIB/S/D:8000/V1:8600/V2:D200/R:FC00/Z:300
```

These linker input command files are stored on the linker diskette. As with the editor, RAM drive will be used extensively. One transfers the compiled, R86, files from the work diskette to the M: drive. Then one transfers the three R86 library files to the M: drive: (1) 87REALS.R86, (2) PASLIB.R86, and (3) TRANCEND.R86. If one does not use the 8087 chip then FPREALS.R86 must be used in place of 87REALS.R86. At this point, one should remove the compiler diskette, and insert the linker diskette. Then one enters Control-C and makes A: the default drive. The input command files save a lot of time since the overlay module names and addresses rarely change during program development. Remember that the linker has been renamed L.CMD for keyboard simplicity. Instead of keying in: "L M:MATRIX,M:CRTLIB, M:TRANCEND,M:87REALS,M:PASLIB/S/E/W" one simply enters: "L M0/F". Without the input command files option, one would have to do eleven links, keying in by hand each time all of the names and addresses in the linker input command files. Instead, one merely executes the eleven commands: (1)"L M0/F", (2)"L M1/F", (3)"L M3/F", (4) "L M5/F", (5) "L M6/F", (6) "L M23/F", (7) "L M19/F", (8) "L M20/F" , (9) "L M21/F", (10) "L M24/F", (11) "L MR/F".

One can further automate the above process by using SUBMIT.CMD to do batch processing of the above eleven commands.

To do this, one must prepare a textfile, with the suffix, SUB. In the linear algebra calculator, this SUB file is called MLINK2.SUB. In this text file, each line of text will be a single command. The eleven command lines of this textfile will be the command lines of the above paragraph. It should be noted that this SUBMIT.CMD code is a very sensitive code. When the MLINK2.SUB was constructed using the Speed Editor, the batch process would not execute. If MLINK2.SUB was constructed using WORDSTAR with no editing changes, it worked fine. Also, some editing such as exchanges gave no problem while deletes did. SUBMIT.CMD seems to perform better under CCPM-86.

The following is the file configuration to do an automatic link. All R86 files including compiled source files and run-time libraries must be on drive M:. The linker diskette, in drive A:,must contain L.CMD and its overlays, SUBMIT.CMD, MLINK2.SUB, and the eleven linker input command files.

With A: as the default drive, one enters "SUBMIT MLINK2" and the linking process will be automatically executed. Once automated, the linking process is not a big hassle. Linking MT+ code is the price one pays for modular compilation. The link will be fairly fast, considering the size of the codes. The fact that the R86 files are being read from RAM drive and the final overlays are being written to RAM drive creates a tremendous savings in time. The linker will create ten overlays: (1) MATRIX.CMD, (2) MATRIX.001, (3) MATRIX.003, (4) MATRIX.005, (5) MATRIX.006, (6) MATRIX.017, (7) MATRIX.013, (8) MATRIX.014, (9) MATRIX.015, (10) MATRIX.018. After they have been created, they

should be transferred to another diskette to be saved and executed as desired. It might be possible to put the linker and its overlays and the KMD files on RAM disk also if one had a larger RAM drive. It does not seem that there is a tremendous loss in time to load the linker overlays and read the command files from a diskette. If the R86 files are read from a diskette and the compiled overlays are wriiten to a diskette, there is a tremendous loss of time. One has to keep in mind that SUBMIT.CMD is sensitive to the selection of drives that have the files that it is batch processing. It seems only experimentation tells what SUBMIT.CMD accepts; it went for the above configuration.

References

1. Brief Instructions for Using Mathlib(Version 6.0), Sandia Laboratories, February 1976.
2. CPM-86 Operating System Manual, Digital Research, Inc., 1983.
3. Jensen, K. and Wirth, N., Pascal User Manual and Report, Springer-Verlag, 1974.
4. Mundie, David A., "An Automatic Metric Conversion Program", Byte Book of Pascal, Byte Publications, pp. 191-192, 1979.
5. Pascal MT+ Manual, Digital Research, Inc., 1983.

```
(* VERSION 0285 *)

PROGRAM MATRIX;

CONST MAXDEG = 20;
      MAXDEGP1 = 21;

TYPE DOMRP1 = 1..MAXDEGP1;
     DOMAIN1 = 1..20;
     MATRIX = ARRAY[DOMAIN1,DOMAIN1] OF REAL;
     LISTI = ARRAY[DOMAIN1] OF INTEGER;
     LISTR = ARRAY[DOMAIN1] OF REAL;
     LISTRP1 = ARRAY[DOMRP1] OF REAL;
     CRTCOMMAND = (ERASEOS,ERASEOL,UP,DOWN,RIGHT,LEFT,LEADIN,TIME,
                   FCOLOR,BCOLOR,REVIDON,REVIDOFF,INTENON,INTENOFF,
                   BLINKON,
                   BLINKOFF);
     SETOFCHAR = SET OF CHAR;
     PTR = ^INTEGER;
     CPMOPERATION = (COLDBOOT,WARMBOOT,CONSTAT,CONIN,CONOUT,LIST,
                     PUNCUT,RDRIN,HOME,SELDSK,SETTRK,SETSEC,SETDMA,
                     DSKREAD,DSKWRITE);
     STRING40 = STRING[40];
     STRING6 = STRING[6];
     STRING80 = STRING[80];


VAR I,J,IONDIM1,IOM1,ROWDIMD: INTEGER;
    SELECT: CHAR;
    OKSET: SETOFCHAR;
    MOIFLAG,IODFLAG,QUITFLAG,SYMFLAG: BOOLEAN;
    IOTA,TA,TVEC: MATRIX;
    IOTB,TEMPTB,XX: MATRIX;
    TEV,TEVR,TEVI: LISTR;
    SBLASTX,SBLASTY: EXTERNAL INTEGER;
    TEMPIOM,TEMPION: INTEGER;
    SODET: REAL;

(* EXTERNAL PROCEDURES AND FUNCTIONS *)

    EXTERNAL PROCEDURE CRTINIT;

    EXTERNAL PROCEDURE CRT(C:CRTCOMMAND);

    EXTERNAL PROCEDURE GOTOXY(X,Y:INTEGER);

    EXTERNAL PROCEDURE PROMPTAT(Y:INTEGER;S:STRING);

    EXTERNAL PROCEDURE CLEARSCREEN;

    EXTERNAL PROCEDURE CLEARIT(I:INTEGER);

    EXTERNAL FUNCTION GETCHAR(OKSET:SETOFCHAR): CHAR;

    EXTERNAL PROCEDURE GETSTRING(VAR S:STRING;MAXLEN: INTEGER);
```

```
    EXTERNAL FUNCTION YES: BOOLEAN;

    EXTERNAL PROCEDURE WAIT;

    EXTERNAL PROCEDURE WHEAD(S:STRING);

    EXTERNAL PROCEDURE INTREAD(VAR K:INTEGER);

    EXTERNAL FUNCTION VALUE(VAR S:STRING; VAR P:INTEGER): REAL;

    EXTERNAL PROCEDURE GETREAL(VAR S:STRING;MAXLEN:INTEGER);

    EXTERNAL [3] PROCEDURE MATRIXIO;

    EXTERNAL [23] PROCEDURE HELP;

    EXTERNAL [19] PROCEDURE TTRDET;

    EXTERNAL [20] PROCEDURE TTSAXB;

    EXTERNAL [21] PROCEDURE TTRPQR;

    EXTERNAL [24] PROCEDURE TTRMAA;

(* END OF EXTERNAL DECLARATIONS *)

  PROCEDURE PREAXB;
  (* SAVES B WHILE THE INVERSE OF A IS COMPUTED *)
  VAR I,J: INTEGER;
  BEGIN (* PREAXB *)
    IF IOBFLAG THEN
      BEGIN
        TEMPICM := ICM;
        FOR I:=1 TO ROWDIMB DO
          FOR J:=1 TO ICM DO
            TEMPTB[I,J] := IOTB[I,J];
        TEMPICM := ROWDIMB;
      END;
    ICM := ICNDIM;
    FOR I:=1 TO ICNDIM DO
      FOR J:=1 TO ICNDIM DO
        IOTB[I,J] := 0.0;
    FOR I:=1 TO ICNDIM DO
      IOTB[I,I] := 1.0;
    TTSAXB;
    IF IOBFLAG THEN
      BEGIN
        ICM := TEMPICM;
        FOR I:=1 TO ROWDIMB DO
          FOR J:=1 TO ICM DO
            IOTB[I,J] := TEMPTB[I,J];
      END;
  END; (* PREAXB *)
```

```
BEGIN(* MATRIX *)
  CRTINIT;
  QUITFLAG := FALSE;
  AOIFLAG := FALSE;
  IOBFLAG := FALSE;
  REPEAT
    CLEARSCREEN;
    WRITELN('LINEAR ALGEBRA CALCULATOR');
    GOTOXY(0,3);
    WRITELN(' ':5,'A  ','REAL POLYNOMIAL ROOT SOLVER');
    WRITELN;
    WRITELN(' ':5,'B  ','EIGENVECTORS OF REAL MATRIX');
    WRITELN;
    WRITELN(' ':5,'C  ','SOLUTION OF NONSINGULAR LINEAR EQUATIONS');
    WRITELN;
    WRITELN(' ':5,'D  ','DETERMINANT OF REAL MATRIX');
    WRITELN;
    WRITELN(' ':5,'E  ','INVERSE OF REAL MATRIX');
    WRITELN;
    WRITELN(' ':5,'F  ','EDITING AND DISPLAY OF INPUT MATRICES');
    WRITELN;
    WRITELN(' ':5,'G  ','DIRECTIONS FOR USE OF CALCULATOR');
    WRITELN;
    WRITELN(' ':5,'Q  ','QUIT');
    WRITELN;
    WRITE(' ':5,'SELECT ONE : ');
    IF AOIFLAG THEN
      BEGIN
        OKSET := ['A'..'F']-['C'];
        IF IOBFLAG THEN
          BEGIN
            IF ROWDIMB=IONDIM THEN
              OKSET := OKSET+['C'];
          END;
      END
    ELSE
      OKSET := ['A','F'];
    OKSET := OKSET+['G','Q'];
    SELECT := GETCHAR(OKSET);
    CLEARSCREEN;
    IF SELECT='Q' THEN
      BEGIN
        PROMPTAT(10,'DO YOU WANT TO ERASE THE EDITED MATRICES? Y/N ')
        IF YES THEN
          QUITFLAG := TRUE;
        CLEARSCREEN;
      END
    ELSE
      CASE SELECT OF
        'A': TTRPOR;
        'B': TTRNVA;
        'C': TTSAXB;
        'D': TTRDET;
        'E': PREAXB;
        'F': MATRIXIO;
```

```
        'G': HELP;
      END;
  UNTIL QUITFLAG;
END. (* MATRIX *)
```

```
(* VERSION 0289 *)

MODULE CRTLIB;


TYPE CRTCOMMAND = (ERASEOS,ERASEOL,UP,DOWN,RIGHT,LEFT,LEADIN,TIME,
                   FCOLOR,BCOLOR,REVIDON,REVIDOFF,INTENON,INTENOFF,
                   BLINKON,
                   BLINKOFF);
     SETOFCHAR = SET OF CHAR;
     PTR = ^INTEGER;
     CPMOPERATION = (COLDBOOT,WARMBOOT,CONSTAT,CONIN,CONOUT,LIST,
                     PUNOUT,RDRIN,HOME,SELDSK,SETTRK,SETSEC,SETDMA,
                     DSKREAD,DSKWRITE);
     COLOR = (BLACK,BLUE,GREEN,CYAN,RED,MAGENTA,BROWN,LGRAY,GRAY,LBLUE,
              LGREEN,LCYAN,LRED,LMAGENTA,YELLOW,WHITE);
     STRING40 = STRING[40];

     CONST BELL = 07;
           RTN = 13;
           BSP = 8;

VAR SBLASTX,SBLASTY: INTEGER;
    CRTINFO: ARRAY[CRTCOMMAND] OF CHAR;
    COLORINFO: ARRAY[COLOR] OF INTEGER;
    PREFIXED: ARRAY[CRTCOMMAND] OF BOOLEAN;
    BDOSVAL: INTEGER; (* GLOBAL VARIABLE FOR BIOS CALLS *)


    EXTERNAL FUNCTION GBDOS86(FUNC:INTEGER; PARM:PTR): INTEGER;


PROCEDURE BIOSCALL(FUNC:CPMOPERATION;CCH:INTEGER);

VAR DESCRIPT: ARRAY[1..5] OF BYTE;
    TBITE: BYTE;
    J: INTEGER;

BEGIN (* BIOSCALL *)
  IF FUNC=CONOUT THEN
    BEGIN
      TBITE := CCH;
      CLRBIT(TBITE,8);
      DESCRIPT[1] := 4;
      DESCRIPT[2] := TBITE;
      DESCRIPT[3] := ' ';
      DESCRIPT[4] := ' ';
      DESCRIPT[5] := ' ';
      BDOSVAL := GBDOS86(50,ADDR(DESCRIPT[1]));
    END;
  IF FUNC=CONIN THEN
    BEGIN
      DESCRIPT[1] := 3;
      FOR J:=2 TO 5 DO
        DESCRIPT[J] := ' ';
```

```
      BDOSVAL := @BDOS86(50,ADDR(DESCRIPT[1]));
    END;
END; (* BIOSCALL *)


PROCEDURE CRTINIT;
VAR OPCRT: CRTCOMMAND;
BEGIN (* CRTINIT *)
  CRTINFO[LEADIN] := CHR(27);
  CRTINFO[ERASEOS] := 'J';
  CRTINFO[ERASEOL] := 'K';
  CRTINFO[RIGHT] := 'C';
  CRTINFO[LEFT] := 'D';
  CRTINFO[UP] := 'A';
  CRTINFO[DOWN] := 'B';
  CRTINFO[TIME] := ' ';
  CRTINFO[FCOLOR] := 'b';
  CRTINFO[BCOLOR] := 'c';
  CRTINFO[REVIDON] := 'p';
  CRTINFO[REVIDOFF] := 'q';
  CRTINFO[INTENON] := 'r';
  CRTINFO[INTENOFF] := 'q';
  CRTINFO[BLINKON] := 's';
  CRTINFO[BLINKOFF] := 't';
  PREFIXED[LEADIN] := FALSE;
  PREFIXED[ERASEOS] := TRUE;
  PREFIXED[ERASEOL] := TRUE;
  PREFIXED[RIGHT] := TRUE;
  PREFIXED[LEFT] := TRUE;
  PREFIXED[UP] := TRUE;
  PREFIXED[DOWN] := TRUE;
  PREFIXED[TIME] := FALSE;
  FOR OPCRT:=FCOLOR TO BLINKOFF DO
    PREFIXED[OPCRT] := TRUE;
END; (* CRTINIT *)


PROCEDURE CRT(C:CRTCOMMAND);
BEGIN (* CRT *)
  IF PREFIXED[C] THEN
    BIOSCALL(CONOUT,ORD(CRTINFO[LEADIN]));
  BIOSCALL(CONOUT,ORD(CRTINFO[C]));
END; (* CRT *)


PROCEDURE GOTOXY(X,Y:INTEGER);
VAR I: INTEGER;
BEGIN  (* GOTOXY *)
  FOR I:=1 TO 100 DO;
  CRT(LEADIN);
  FOR I:=1 TO 100 DO;
  BIOSCALL(CONOUT,ORD('Y'));
  FOR I:=1 TO 100 DO;
  BIOSCALL(CONOUT,Y+32);
  FOR I:=1 TO 100 DO;
```

```
  BIOSCALL(CONOUT,X+32);
  SBLASTX := X;
  SBLASTY := Y;
  FOR I:=1 TO 100 DO;
END; (* GOTOXY *)


PROCEDURE PROMPTAT(Y:INTEGER;S:STRING);
VAR J: INTEGER;
BEGIN (* PROMPTAT *)
  GOTOXY(0,Y);
  WRITE(S);
  CRT(ERASEOL);
  FOR J:=1 TO 4 DO
    CRT(TIME);
END; (* PROMPTAT *)

PROCEDURE CLEARSCREEN;
VAR J: INTEGER;
BEGIN (* CLEARSCREEN *)
  GOTOXY(0,0);
  CRT(ERASEOS);
  FOR J:=1 TO 4 DO
    CRT(TIME);
END; (* CLEARSCREEN *)

PROCEDURE CLEARIT(I:INTEGER);
VAR J: INTEGER;
BEGIN (* CLEARIT *)
  GOTOXY(0,I);
  CRT(ERASEOS);
  FOR J:=1 TO 4 DO
    CRT(TIME);
END; (* CLEARIT *)

FUNCTION GETCHAR(OKSET:SETOFCHAR): CHAR;
VAR CH: CHAR;
    OCH: INTEGER;
    GOOD: BOOLEAN;
BEGIN (* GETCHAR *)
  REPEAT
    BIOSCALL(CONIN,0); (* 0 IS A DUMMY VARIABLE *)
    CH := CHR(DDOSVAL);
    OCH := ORD(CH);
    IF OCH>96 THEN
      IF OCH<123 THEN
        CH := CHR(OCH-32);
    GOOD := CH IN OKSET;
    IF NOT GOOD THEN
      WRITE(CHR(7))
    ELSE
      IF CH IN [' '..'}'] THEN
        WRITE(CH);
  UNTIL GOOD;
  GETCHAR := CH;
```

```
END; (* GETCHAR *)


PROCEDURE GETSTRING(VAR S: STRING; MAXLEN: INTEGER);
VAR S1: STRING[1];
    STEMP: STRING;
    OKSET: SET OF CHAR;
BEGIN (* GETSTRING *)
  OKSET := [' '..'}'];
  S1 := ' ';
  STEMP := '';
  REPEAT
    IF LENGTH(STEMP) = 0 THEN
      S1[1] := GETCHAR(OKSET )
    ELSE
      IF LENGTH(STEMP)=MAXLEN THEN
        S1[1] := GETCHAR([CHR(RTN),CHR(
                 BSP)])
    ELSE
      S1[1] := GETCHAR(OKSET + [CHR(RTN),CHR(BSP)]);
    IF S1[1] IN OKSET THEN
      STEMP := CONCAT(STEMP,S1)
    ELSE
      IF S1[1]=CHR(BSP) THEN
        BEGIN
          CRT(LEFT);
          WRITE(' ');
          CRT(LEFT);
          DELETE(STEMP,LENGTH(STEMP),1);
        END;
  UNTIL S1[1] = CHR(RTN);
  IF LENGTH(STEMP) <> 0 THEN
    S := STEMP
  ELSE
    WRITE(S);
END; (* GETSTRING *)

FUNCTION YES: BOOLEAN;
BEGIN (* YES *)
  YES := GETCHAR(['Y','N']) IN ['Y'];
END; (* YES *)



PROCEDURE WAIT;
BEGIN (* WAIT *)
  CLEARIT(5);
  PROMPTAT(10,'PLEASE WAIT...');
END; (* WAIT *)

PROCEDURE WHEAD(A:STRING);
VAR I: INTEGER;
BEGIN (* WHEAD *)
  CLEARSCREEN;
  I := (80-LENGTH(A)) DIV 2;
```

```
  GOTOXY(I,0);
  WRITELN(A);
  GOTOXY(I,1);
  FOR I:=1 TO LENGTH(A) DO
    IF A[I]=' ' THEN
      WRITE(' ')
    ELSE
      WRITE('-');
  WRITELN;
END;(*WHEAD*)

PROCEDURE INTREAD(VAR K:INTEGER);
TYPE STRING6 = STRING[6];
VAR S: STRING6;

  PROCEDURE GETISTRING(VAR S:STRING6;MAXLEN:INTEGER);
  VAR S1: STRING[1];
      STEMP,TTEMP: STRING6;
      OKSET,OKAYSET: SET OF CHAR;
      FLAG1,FLAG2,NFLAG: BOOLEAN;
      I,L,MAX,T1,T2,T3,T4,T5: INTEGER;

    PROCEDURE CHECKINT;
    VAR I: INTEGER;
    BEGIN(* CHECKINT *)
      TTEMP := STEMP;
      IF FLAG2 THEN
        DELETE(TTEMP,1,1);
      IF LENGTH(TTEMP)<5 THEN
        FLAG1 := FALSE
      ELSE
        BEGIN
          T1 := ORD(TTEMP[1]);
          T2 := ORD(TTEMP[2]);
          T3 := ORD(TTEMP[3]);
          T4 := ORD(TTEMP[4]);
          T5 := ORD(TTEMP[5]);
          IF T1<=51 THEN
            IF T1=51 THEN
              IF T2<=50 THEN
                IF T2=50 THEN
                  IF T3<=55 THEN
                    IF T3=55 THEN
                      IF T4<=54 THEN
                        IF T4=54 THEN
                          BEGIN
                            IF NFLAG THEN
                              BEGIN
                                IF T5<=56 THEN
                                  FLAG1 := FALSE
                                ELSE
                                  FLAG1 := TRUE
                              END
                            ELSE
                              BEGIN
```

```
                                    IF T5<=55 THEN
                                      FLAG1 := FALSE
                                    ELSE
                                      FLAG1 := TRUE
                                  END
                              END
          ELSE
            FLAG1 := FALSE
          ELSE
            FLAG1 := TRUE
          ELSE
            FLAG1 := FALSE
          ELSE
            FLAG1 := TRUE
          ELSE
            FLAG1 := FALSE
          ELSE
            FLAG1 := TRUE
          ELSE
            FLAG1 := FALSE
          ELSE
            FLAG1 := TRUE;
        END;
      IF FLAG1=TRUE THEN
        BEGIN
          FLAG1 := TRUE;
          L := LENGTH(STEMP);
          FOR I:=1 TO L DO
            CRT(LEFT);
          FOR I:=1 TO L DO
            WRITE(' ');
          FOR I:=1 TO L DO
            CRT(LEFT);
          FOR I:=1 TO L DO
            WRITE(CHR(BELL));
        END;
    END;(* CHECKINT *)

  BEGIN(* GETISTRING *)
    REPEAT
      OKSET := ['0'..'9'];
      S1 := ' ';
      STEMP := '';
      NFLAG := FALSE;
      REPEAT
        IF LENGTH(STEMP) = 0 THEN
          BEGIN
            MAX := MAXLEN-1;
            FLAG2 := FALSE;
            OKAYSET := OKSET+['+','-'];
            S1[1] := GETCHAR(OKAYSET);
            IF S1[1] IN (['+','-']) THEN
              BEGIN
                IF S1[1]='-' THEN
                  NFLAG := FALSE;
```

```
              MAX := MAX+1;
              FLAG2 := TRUE;
            END;
        END
      ELSE
        IF LENGTH(STEMP)=MAX THEN
          S1[1] := GETCHAR([CHR(RTN),CHR
                (BSP)])
      ELSE
        BEGIN
          IF (LENGTH(STEMP)=1)  AND FLAG2
            THEN
            S1[1] := GETCHAR(OKSET+[CHR(BSP)])
          ELSE
            S1[1] := GETCHAR(OKSET + [CHR(RTN),CHR(BSP)]);
        END;
      IF S1[1] IN (OKSET+['+','-'])  THEN
        STEMP := CONCAT(STEMP,S1
                 )
      ELSE
        IF S1[1]=CHR(BSP) THEN
          BEGIN
            CRT(LEFT);
            WRITE(' ');
            CRT(LEFT);
            DELETE(STEMP,LENGTH(STEMP),1);
          END;
    UNTIL S1[1] = CHR(RTN);
    CHECKINT;
  UNTIL NOT FLAG1;
  S := STEMP;
END;(* GETISTRING *)

PROCEDURE STRTOINT(VAR S:STRING6;VAR K:INTEGER);
CONST Z = 48;
VAR STEMP: STRING6;
    FLAGP: BOOLEAN;
    I,L: INTEGER;

BEGIN(* STRTOINT *)
  STEMP := S;
  IF STEMP[1]='-' THEN
    FLAGP := FALSE
  ELSE
    FLAGP := TRUE;
  IF (NOT FLAGP) OR (STEMP[1]='+') THEN
    DELETE(STEMP,1,1);
  L := LENGTH(STEMP);
  K := 0;
  FOR I:=1 TO L DO
    K := 10*K+ORD(STEMP[I])-Z;
  IF FLAGP=FALSE THEN
    K := -K;
END;(* STRTOINT *)
```

```
BEGIN(* INTREAD *)
  GETISTRING(S,6);
  STRTOINT(S,K);
END;(* INTREAD *)


FUNCTION VALUE(VAR S:STRING; VAR P:INTEGER): REAL;
CONST
      LIMIT = 1.0E+16;
      Z = 48; (* ORD(0) *)
VAR A,Y: REAL;
    E,I,J,P2: INTEGER;
    NEG,NEGEXP,CTL: BOOLEAN;
    DIGITS: SET OF CHAR;

  FUNCTION POWRTEN(EX:INTEGER): REAL;
  VAR I: INTEGER;
      T: REAL;
  BEGIN (* POWRTEN *)
    I := 0;
    T := 1.0;
    REPEAT
      IF ODD(EX) THEN
        CASE I OF
          0: T := T*1.0E1;
          1: T := T*1.0E2;
          2: T := T*1.0E4;
          3: T := T*1.0E8;
          4: T := T*1.0E16;
          5: T := T*1.0E32;
          6: T := T*1.0E64;
          7: T := T*1.0E128;
          8: T := T*1.0E256;
        END;
      EX := EX DIV 2;
      I := I+1;
    UNTIL EX=0;
    POWRTEN := T;
  END; (* POWRTEN *)

BEGIN (* VALUE *)
  I := 1;
  P := 0;
  P2 := 0;
  CTL := FALSE;
  DIGITS := ['0'..'9'];
  S := CONCAT(S,'%');(*SAFETY CHARACTER *)
  A := 0;
  E := 0;
  NEG := (S[I]='-');
  WHILE S[I]=' ' DO
    I := I+1;
  IF (S[I]='+') OR NEG THEN
    I := I+1;
```

```
WHILE S[I] IN DIGITS DO
  BEGIN
    IF S[I]='0' THEN
      P2 := P2+1
    ELSE
      BEGIN
        P := P+P2+1;
        P2 := 0;
        GTL := TRUE;
      END;
    IF A<LIMIT THEN
      A := 10*A+ORD(S[I])-Z
    ELSE
      E := E+1;
    I := I+1;
  END;
IF S[I]='.' THEN
  BEGIN
    P := P+P2;
    I := I+1;
    IF NOT (S[I] IN DIGITS) THEN
      BEGIN
        INSERT('0',S,I);
        I := I+1;
      END;
  END;
P2 := 0;
WHILE S[I]='0' DO
  BEGIN
    P2 := P2+1;
    IF A<LIMIT THEN
      BEGIN
        A := 10*A+ORD(S[I])-Z;
        E := E-1;
      END;
    I := I+1;
  END;
IF GTL THEN
  P := P+P2;
WHILE S[I] IN DIGITS DO
  BEGIN
    P := P+1;
    IF A<LIMIT THEN
      BEGIN
        A := 10*A+ORD(S[I])-Z;
        E := E-1;
      END;
    I := I+1;
  END;
IF S[I] IN ['E','E'] THEN
  BEGIN
    I := I+1;
    J := 0;
    NEGEXP := (S[I]='-');
    IF (S[I]='+') OR NEGEXP THEN
```

```
        I := I+1;
      WHILE S[I] IN DIGITS DO
        BEGIN
          IF J<LIMIT THEN
            J := 10*J+ORD(S[I])-Z;
          I := I+1;
        END;
      IF NEGEXP THEN
        E := E-J
      ELSE
        E := E+J;
    END;
  Y := A;
  IF NEG THEN
    Y := -Y;
  IF E<0 THEN
    VALUE := Y/POWRTEN(-E)
  ELSE
    IF E<>0 THEN
      VALUE :=
              Y*POWRTEN(E)
  ELSE
    VALUE := Y;
  IF ((NOT NEG) AND (Y=0.0)) THEN
    VALUE := -Y;
  WHILE S[I]=' ' DO
    I := I+1;
  S := COPY(S,I,LENGTH(S)-I);
END; (* VALUE *)

PROCEDURE GETREAL(VAR S:STRING;MAXLEN:INTEGER);
VAR S1,S2: STRING[1];
    STEMP: STRING[80];
    INTSET,ALPSET,SIGSET,OKSET,MISCSET: SET OF CHAR;
    LOC: INTEGER;
    KAR,CHOICE,NTOER: INTEGER;
    PERFLAG,EXPFLAG: BOOLEAN;
BEGIN  (* GETREAL *)
  ALPSET := ['A'..'Z'];
  INTSET := ['0'..'9'];
  SIGSET := ['+','-'];
  MISCSET := ['E','.'];
  EXPFLAG := FALSE;
  STEMP := '';
  S1 := ' ';
  S2 := ' ';
  PERFLAG := FALSE;
  KAR := 0;
  REPEAT
    LOC := LENGTH(STEMP);
    IF LENGTH(STEMP)=0 THEN
      BEGIN
        OKSET := MISCSET+INTSET+SIGSET;
        S1[1] := GETCHAR(OKSET+[CHR(RTN)]);
      END
```

```
ELSE
  IF LENGTH(STEMP)=MAXLEN THEN
    BEGIN
      OKSET := [CHR(RTN)];
      S1[1] := GETCHAR(OKSET+[CHR(BSP)]);
    END
ELSE
  BEGIN
    IF NOT EXPFLAG THEN
      BEGIN
        LOC := LENGTH(STEMP);
        S2[1] := STEMP[LOC];
        IF S2[1] IN SIGSET THEN
          CHOICE := 1
        ELSE
          IF S2[1] IN INTSET THEN
            CHOICE := 2
        ELSE
          IF S2[1]='.' THEN
            CHOICE := 3;
        CASE CHOICE OF
          1: OKSET := MISCSET+INTSET;
          2: OKSET := MISCSET+INTSET+[CHR(RTN)];
          3: OKSET := MISCSET+INTSET+[CHR(RTN)];
        END;
      END
    ELSE
      BEGIN
        LOC := LENGTH(STEMP);
        S2[1] := STEMP[LOC];
        IF S2[1] IN SIGSET THEN
          CHOICE := 1
        ELSE
          IF S2[1] IN INTSET THEN
            CHOICE := 2
        ELSE
          IF S2[1]='E' THEN
            CHOICE := 3;
        CASE CHOICE OF
          1: OKSET := INTSET;
          2: OKSET := INTSET+[CHR(RTN)];
          3: OKSET := SIGSET+INTSET;
        END;
      END;
    S1[1] := GETCHAR(OKSET+[CHR(BSP)]);
  END;
IF S1[1] IN (OKSET-[CHR(RTN)]) THEN
  BEGIN
    STEMP := CONCAT(STEMP,S1);
    IF S1[1]='E' THEN
      BEGIN
        EXPFLAG := TRUE;
        MISCSET := MISCSET-['E'];
      END
    ELSE
```

```
            IF S1[1]='.' THEN
              BEGIN
                PERFLAG := TRUE;
                MISCSET := MISCSET-['.'];
              END;
        END
      ELSE
        IF S1[1]=CHR(BSP) THEN
          BEGIN
            LOC := LENGTH(STEMP);
            IF STEMP[LOC]='.' THEN
              PERFLAG := FALSE
            ELSE
              IF STEMP[LOC]='E' THEN
                EXPFLAG := FALSE;
            CRT(LEFT);
            WRITE(' ');
            CRT(LEFT);
            DELETE(STEMP,LOC,1);
          END;
      IF NOT EXPFLAG THEN
        MISCSET := MISCSET+['E'];
      IF NOT PERFLAG THEN
        MISCSET := MISCSET+['.'];
    UNTIL S1[1]=CHR(RTN);
    IF LENGTH(STEMP)<>0 THEN
      BEGIN
        S := STEMP;
        NTOBR := 23-LENGTH(S);
        IF NTOBR>0 THEN
          WRITE(' ':NTOBR);
      END;
END; (* GETREAL *)

PROCEDURE SPACEBAR;
VAR CH: CHAR;
BEGIN (* SPACEBAR *)
  WRITELN;
  WRITE('PRESS SPACEBAR ');
  CH := GETCHAR([' ']);
  WRITELN;
END; (* SPACEBAR *)

(* THE FOLLOWING ROUTINES ARE FOR COLOR MONITORS *)


(* PROCEDURE INITCOLOR;
   VAR SHADE:COLOR;
       I:INTEGER;
   BEGIN
     I:=0;
     FOR SHADE:=BLACK TO WHITE DO
     BEGIN
       COLORINFO[SHADE]:=I;
```

```
      I:=I+1;
    END;
  END; *)

(* PROCEDURE ALTCOLOR(C:COLOR);
  VAR J:INTEGER;
  BEGIN
    CRT(BCOLOR);
    FOR J:=1 TO 100 DO;
    BIOSCALL(CONOUT,COLORINFO[C]);
    FOR J:=1 TO 100 DO;
  END; *)

(* PROCEDURE PAINT(X,Y,WIDTH,DEPTH:INTEGER;SHADE:COLOR);
  VAR J:INTEGER;
  BEGIN
    GOTOXY(X,Y);
    ALTCOLOR(SHADE);
    FOR J:=1 TO DEPTH DO
    BEGIN
      WRITE(' ':WIDTH);
      GOTOXY(X,Y+J);
    END;
    FOR J:=1 TO 100 DO;
  END; *)

MODEND.
```

```
(* VERSION 0286 *)
MODULE OVERLAY1;
(* MODULE LINSYS  *)

TYPE DOMAIN1 = 1..20;
     MATRIX = ARRAY[DOMAIN1,DOMAIN1] OF REAL;
     LISTI = ARRAY[DOMAIN1] OF INTEGER;
     LISTR = ARRAY[DOMAIN1] OF REAL;

EXTERNAL PROCEDURE SPACEBAR;

PROCEDURE RLUD(ND,N:INTEGER;VAR KER:INTEGER;VAR ALU:MATRIX;VAR JN:
                LISTI;VAR SCALE:LISTR);

VAR I,IND,IP1,IC,J,K,NN: INTEGER;
    BIG,EL,PIVOT,ROWNRM,T: REAL;
    CH: CHAR;


  FUNCTION AMAX1(A,B:REAL): REAL;
  BEGIN (* AMAX1 *)
    IF A<B THEN
      AMAX1 := B
    ELSE
      AMAX1 := A;
  END; (* AMAX1 *)

BEGIN (* RLUD *)
  NN := N;
(* COMPUTE SCALE[I]=1.0/INFINITY NORM OF ROW[I] OF A *)
  FOR I:=1 TO NN DO
    BEGIN
      ROWNRM := 0.0;
      FOR J:=1 TO NN DO
        ROWNRM := AMAX1(ROWNRM,ABS(ALU[I,J]));
      IF ROWNRM=0.0 THEN
        ROWNRM := 1.0;
      SCALE[I] := 1.0/ROWNRM;
    END;
(* LU DECOMPOSITION BY GAUSSIAN ELIMINATION.  L HAS UNIT DIAGONAL.
(* EXPLICIT ROW INTERCHANGE WITH IMPLICIT EQUILIBRATION IS USED *)
  IS := 1;
  FOR I:=1 TO NN DO
    BEGIN
      WRITE('...');
      IND := I;
      IF I<>NN THEN
        BEGIN
          BIG := 0.0;
          FOR K:=I TO NN DO
            BEGIN
              T := SCALE[K]*ABS(ALU[K,I]);
              IF T>BIG THEN
                BEGIN
                  IND := K;
```

```
                BIG := T;
              END;
          END;
        IF BIG<>0.0 THEN
          BEGIN
            IF IND<>I THEN
              BEGIN
                FOR J:=I TO NN DO
                  BEGIN
                    T := ALU[IND,J];
                    ALU[IND,J] := ALU[I,J];
                    ALU[I,J] := T;
                  END;
                SCALE[IND] := SCALE[I];
                IS := -IS;
              END;
            IPl := I+1;
            PIVOT := ALU[I,I];
            FOR K:=IPl TO NN DO
              BEGIN
                EL := -ALU[K,I]/PIVOT;
                ALU[K,I] := -EL;
                IF EL<>0.0 THEN
                  FOR J:=IPl TO NN DO
                    ALU[K,J] := ALU[K,J]+EL*ALU[I,J];
              END;
          END;
      END;
    IF ALU[I,I]=0.0 THEN
      IS := 0;
    JN[I] := IND;
  END;
 JN[NN] := IS;
 KER := 0;
END; (* RLUD *)

PROCEDURE RFBS(ND,N:INTEGER;VAR KER:INTEGER;VAR ALU:MATRIX;VAR JN:
               LISTI;VAR X:LISTR);

VAR I,K,KPl,L,LPl,NM1,NN: INTEGER;
    Z: REAL;
    CH: CHAR;
BEGIN  (* RFBS *)
  NN := N;
  IF JN[NN]=0 THEN
    BEGIN
      KER := 3;
      WRITELN;
      WRITELN('IN RFBS, THE TRIANGULAR FACTOR U OF A ');
      WRITELN(' IS SINGULAR. A UNIQUE SOLUTION DOES ');
      WRITELN(' NOT EXIST.');
      SPACEBAR;
      EXIT;
    END;
  KER := 0;
```

```
    NM1 := NN-1;
    IF NM1<>0 THEN
      BEGIN
  (* SOLVE LY=B (FORWARD SUBSTITUTION) *)
        FOR L:=1 TO NM1 DO
          BEGIN
            WRITE('...');
            K := JN[L];
            Z := X[K];
            X[K] := X[L];
            X[L] := Z;
            LP1 := L+1;
            FOR K:=LP1 TO NN DO
              X[K] := X[K]-ALU[K,L]*Z;
          END;
  (* SOLVE UX=Y (BACKWARD SUBSTITUTION) *)
        FOR I:=1 TO NM1 DO
          BEGIN
            WRITE('...');
            K := NN-I;
            KP1 := K+1;
            X[KP1] := X[KP1]/ALU[KP1,KP1];
            Z := -X[KP1];
            FOR L:=1 TO K DO
              X[L] := X[L]+ALU[L,KP1]*Z;
          END;
      END;
    X[1] := X[1]/ALU[1,1];
  END;  (* RFBS *)


  ICDEMO.
```

```
(* VERSION 0285 *)
MODULE OVERLAY3;

CONST MAXDEG = 20;
      MAXDEGP1=21;

TYPE DOMAIN1 = 1..20;
     DOMRP1=1..MAXDEGP1;
     LISTRP1=ARRAY[DOMRP1] OF REAL;
     MATRIX = ARRAY[DOMAIN1,DOMAIN1] OF REAL;
     LISTI = ARRAY[DOMAIN1] OF INTEGER;
     LISTR = ARRAY[DOMAIN1] OF REAL;
     CRTCOMMAND = (ERASEOS,ERASEOL,UP,DOWN,RIGHT,LEFT,LEADIN,TIME,
                   FCOLOR,BCOLOR,REVIDON,REVIDOFF,INTENON,INTENOFF,
                   BLINKON,
                   BLINKOFF);
     SETOFCHAR = SET OF CHAR;
     PTR = ^INTEGER;
     CPMOPERATION = (COLDBOOT,WARMBOOT,CONSTAT,CONIN,CONOUT,LIST,
                     PUNOUT,RDRIN,HOME,SELDSK,SETTRK,SETSEC,SETDMA,
                     DSKREAD,DSKWRITE);
     STRING40 = STRING[40];
     STRING6 = STRING[6];
     STRING80 = STRING[80];



VAR ACIFLAG,ICDFLAG: EXTERNAL BOOLEAN;
    IOM,ICNDIM,ROWDIMB: EXTERNAL INTEGER;
    IOTA,TVEC: EXTERNAL MATRIX;
    IOTB,XX: EXTERNAL MATRIX;
    TEV,TEVR,TEVI: EXTERNAL LISTR;
    GSDET: EXTERNAL REAL;

(* EXTERNAL PROCEDURES AND FUNCTIONS *)

    EXTERNAL PROCEDURE CRTINIT;

    EXTERNAL PROCEDURE CRT(C:CRTCOMMAND);

    EXTERNAL PROCEDURE GOTOXY(X,Y:INTEGER);

    EXTERNAL PROCEDURE PROMPTAT(Y:INTEGER;S:STRING);

    EXTERNAL PROCEDURE CLEARSCREEN;

    EXTERNAL PROCEDURE CLEARIT(I:INTEGER);

    EXTERNAL FUNCTION GETCHAR(OKSET:SETOFCHAR): CHAR;

    EXTERNAL PROCEDURE GETSTRING(VAR S:STRING;MAXLEN: INTEGER);

    EXTERNAL FUNCTION YES: BOOLEAN;

    EXTERNAL PROCEDURE WAIT;
```

```
    EXTERNAL PROCEDURE WHEAD;

    EXTERNAL PROCEDURE INTREAD(VAR K:INTEGER);

    EXTERNAL FUNCTION VALUE(VAR S:STRING; VAR P:INTEGER): REAL;

    EXTERNAL PROCEDURE GETREAL(VAR S:STRING;MAXLEN:INTEGER);

    EXTERNAL PROCEDURE SPACEBAR;


PROCEDURE GETCOLUMN(NROW,J:INTEGER;VAR A:MATRIX);
VAR II,K,LL,P4: INTEGER;
    SREEL: STRING;
    FLAG: BOOLEAN;

  PROCEDURE COLDISPLAY;
  VAR IC: INTEGER;
  BEGIN (* COLDISPLAY *)
    CLEARIT(2);
    GOTOXY(0,2);
    FOR IC:=1 TO NROW DO
      BEGIN
        K := IC+1;
        GOTOXY(0,K);
        WRITE('A[',IC,',',J,']=');
        GOTOXY(9,K);
        WRITE(A[IC,J]);
      END;
    PROMPTAT(23,'IS THIS CORRECT? Y/N ');
  END; (* COLDISPLAY *)

BEGIN (* GETCOLUMN *)
  FLAG := FALSE;
  REPEAT
    CLEARIT(2);
    COLDISPLAY;
    IF NOT YES THEN
      BEGIN
        PROMPTAT(23,'IF NO CHANGE - PRESS RETURN ');
        FOR LL:=1 TO NROW DO
          BEGIN
            K := LL+1;
            GOTOXY(9,K);
            IF A[LL,J]>=0 THEN
              WRITE(' ');
            SREEL := '';
            GETREAL(SREEL,21);
            IF LENGTH(SREEL)>0 THEN
              A[LL,J] := VALUE(SREEL,P4);
          END;
      END
    ELSE
      FLAG := TRUE;
```

```
  UNTIL FLAG;
END; (* GETCOLUMN *)

PROCEDURE GETPOLCOF(NROW:INTEGER;VAR L:LISTRP1);
VAR I,J,NEWI,NRODIV2,P4: INTEGER;
    SREEL: STRING;
    FLAG: BOOLEAN;

  PROCEDURE COLDISPLAY;
  VAR I: INTEGER;
  BEGIN (* COLDISPLAY *)
    CLEARIT(2);
    GOTOXY(0,2);
    NRODIV2 := NROW DIV 2;
    FOR I:=1 TO NRODIV2 DO
      BEGIN
        GOTOXY(0,2*I);
        WRITE('COEF OF X');
        CRT(UP);
        WRITE(NROW-I);
        CRT(DOWN);
        WRITE('=');
        GOTOXY(12,2*I);
        WRITE(L[I]);
      END;
    FOR I:=NRODIV2+1 TO NROW DO
      BEGIN
        NEWI := I-NRODIV2;
        GOTOXY(40,2*NEWI);
        WRITE('COEF OF X');
        CRT(UP);
        WRITE(NROW-I);
        CRT(DOWN);
        WRITE('=');
        GOTOXY(52,2*NEWI);
        WRITE(L[I]);
      END;
    PROMPTAT(23,'IS THIS CORRECT? Y/N ');
  END; (* COLDISPLAY *)

BEGIN (* GETPOLCOF *)
  FLAG := FALSE;
  REPEAT
    CLEARIT(2);
    COLDISPLAY;
    IF NOT YES THEN
      BEGIN
        PROMPTAT(23,'IF NO CHANGE - PRESS RETURN ');
        FOR I:=1 TO NRODIV2 DO
          BEGIN
            GOTOXY(12,2*I);
            IF L[I]>=0 THEN
              WRITE(' ');
            SREEL := '';
            GETREAL(SREEL,21);
```

```
            IF LENGTH(SREEL)<>0 THEN
              L[I] := VALUE(SREEL,P4);
          END;
        FOR I:=NRODIV2+1 TO NROW DO
          BEGIN
            NEWI := I-NRODIV2;
            GOTOXY(52,2*NEWI);
            IF L[I]>=0 THEN
              WRITE(' ');
            SREEL := '';
            GETREAL(SREEL,21);
            IF LENGTH(SREEL)<>0 THEN
              L[I] := VALUE(SREEL,P4);
          END;
      END
    ELSE
      FLAG := TRUE;
  UNTIL FLAG;
END; (* GETPOLCOF *)



PROCEDURE PMEN(I:INTEGER;C:CHAR;S:STRING);
BEGIN (* PMEN *)
  GOTOXY(0,I);
  WRITELN(C:3,' ':3,S);
END; (* PMEN *)

PROCEDURE HEAD(A:STRING;J:INTEGER);
VAR I: INTEGER;
BEGIN (* HEAD *)
  I := (80-LENGTH(A)) DIV 2;
  GOTOXY(I,J);
  WRITELN(A);
END; (* HEAD *)

PROCEDURE MATRIXIC;
VAR QUITFLAG: BOOLEAN;
    CHOICE: CHAR;
    CHSET: SETOFCHAR;

  PROCEDURE EDIT;
  VAR EDQUIT: BOOLEAN;
      EDCHOICE: CHAR;

    PROCEDURE AENTER(AORB:CHAR);
    VAR I,J,COLDIM,ROWDIM: INTEGER;
        MANAME: STRING[3];
    BEGIN (* AENTER *)
      CLEARIT(1);
      CASE AORB OF
        'A': BEGIN
               AOIFLAG := TRUE;
               REPEAT
```

```
            PROMPTAT(2,'ENTER DIMENSION OF SQUARE MATRIX? ');
            INTREAD(ICNDIM);
          UNTIL ((IONDIM>0) AND (ICNDIM<21));
          ROWDIM := ICNDIM;
          COLDIM := ICNDIM;
          MANAME:='A';
        END;
    'B': BEGIN
          IOIFLAG := TRUE;
          REPEAT
            PROMPTAT(2,'ENTER ROW DIMENSION OF B ? ');
            INTREAD(ROWDIMB);
          UNTIL ((ROWDIMB>0) AND (ROWDIMB<21));
          REPEAT
            CLEARIT(1);
            PROMPTAT(2,'ENTER THE COLUMN DIM OF B? ');
            INTREAD(ICM);
          UNTIL ((ICM>0) AND (ICM<21));
          COLDIM := ICM;
          ROWDIM := ROWDIMB;
          MANAME:='B';
        END;
  END;
  FOR I:=1 TO ROWDIM DO
    FOR J:=1 TO COLDIM DO
      CASE AORB OF
        'A': IOTA[I,J] := 0.0;
        'B': IOTB[I,J] := 0.0;
      END;
  FOR J:=1 TO COLDIM DO
    BEGIN
      CLEARIT(1);
      WRITELN('ENTER COLUMN ',J,' OF MATRIX ',MANAME);
      CASE AORB OF
        'A': GETCOLUMN(ROWDIM,J,IOTA);
        'B': GETCOLUMN(ROWDIM,J,IOTB);
      END;
    END;
  CLEARIT(1);
END; (* AENTER *)


PROCEDURE AEDIT(AORB: CHAR);
VAR ACHOICE: CHAR;
    AQUIT,EFLAG: BOOLEAN;
    I,J,COLDIM,ROWDIM: INTEGER;
    MANAME: STRING[3];


BEGIN (* AEDIT *)
  CASE AORB OF
    'A': BEGIN
          EFLAG := AOIFLAG;
          MANAME := 'A';
          ROWDIM := IONDIM;
```

```
            COLDIM := IONDIM;
          END;
      'B': BEGIN
            EFLAG := IOBFLAG;
            MANAME := 'B';
            ROWDIM := ROWDIMB;
            COLDIM := IOM;
          END;
    END;

    IF NOT EFLAG THEN
      AENTER(AORB)
    ELSE
      BEGIN
        AQUIT := FALSE;
        REPEAT
          CLEARIT(1);
          PMEN(2,'A','EDIT CURRENT MATRIX ');
          PMEN(4,'B','CREATE A NEW MATRIX ');
          PMEN(6,'Q','QUIT');
          GOTOXY(2,8);
          WRITE('SELECT ONE : ');
          OKSET := ['A','B','Q'];
          ACHOICE := GETCHAR(OKSET);
          CASE ACHOICE OF
            'A': FOR I:=1 TO COLDIM DO
                   BEGIN
                     CLEARIT(1);
                     WRITELN('ENTER COLUMN ',I,' OF MATRIX ',MANA
                     CASE AORB OF
                       'A': GETCOLUMN(ROWDIM,I,IOTA);
                       'B': GETCOLUMN(ROWDIM,I,IOTB);
                     END;
                   END;
            'B': BEGIN
                   CLEARIT(1);
                   GOTOXY(0,2);
                   WRITE('DO YOU WANT TO ERASE CURRENT MATRIX ',
                         '? Y/N ');
                   CRT(ERASEOL);
                   FOR I:=1 TO 100 DO CRT(TIME);
                   IF YES THEN
                     AENTER(AORB);
                 END;
            'Q': AQUIT := TRUE;
          END;
        UNTIL AQUIT;
      END;
  END; (* AEDIT *)


BEGIN (* EDIT *)
  CLEARSCREEN;
  HEAD('MATRIX EDITOR',0);
  EDQUIT := FALSE;
```

```
  REPEAT
    CLEARIT(1);
    PMEN(2,'A','EDIT MATRIX A');
    PMEN(4,'B','EDIT MATRIX B');
    PMEN(6,'Q','QUIT');
    GOTOXY(2,8);
    WRITE('SELECT ONE -: ');
    OKSET := ['A','B','Q'];
    EDCHOICE := GETCHAR(OKSET);
    CASE EDCHOICE OF
      'A': AEDIT('A');
      'B': AEDIT('B');
      'Q': EDQUIT := TRUE;
    END;
  UNTIL EDQUIT;
END; (* EDIT *)

PROCEDURE DISPLAY;
VAR DISQUIT: BOOLEAN;
    DISCHOICE: CHAR;

  PROCEDURE DDISPLA(AORB:CHAR);
  VAR I,J,COLDIM,ROWDIM: INTEGER;
      CH: CHAR;
      CFLAG: BOOLEAN;
      MANAME: STRING[3];
  BEGIN (* DDISPLA *)
    CLEARIT(1);
    CASE AORB OF
      'A': BEGIN
             CFLAG := AOKFLAG;
             MANAME := 'A';
             COLDIM := ICNDIM;
             ROWDIM := ICNDIM;
           END;
      'B': BEGIN
             CFLAG := BOKFLAG;
             MANAME := 'B';
             COLDIM := ICM;
             ROWDIM := ROWDIMB;
           END;
    END;
    IF NOT CFLAG THEN
      BEGIN
        GOTOXY(0,3);
        WRITE('THERE IS NO MATRIX ',MANAME,' YET ');
        CRT(ERASEOL);
        FOR I:=1 TO 100 DO CRT(TIME);
        SPACEBAR;
      END
    ELSE
      BEGIN
        WRITELN('THE COLUMNS OF MATRIX ',MANAME,' ARE: ');
        FOR J:=1 TO COLDIM DO
          BEGIN
```

```
            WRITELN('COLUMN ',J,' OF ',MANAME,' IS :');
            FOR I:=1 TO ROWDIM DO
            BEGIN
              WRITE(MANAME,'[',I,',',J,']=');
              CASE AORB OF
                'A': WRITELN(ICTA[I,J]);
                'B': WRITELN(ICTB[I,J]);
              END;
            END;
            SPACEBAR;
          END;
      END;
  END; (* DDISPLA *)

  PROCEDURE HCOPY(AORB:CHAR);
  VAR CH: CHAR;
      CFLAG: BOOLEAN;
      MANAME: STRING[3];
      CNUM,I: INTEGER;
  BEGIN (* HCOPY *)
    CASE AORB OF
      'A': BEGIN
             CFLAG := AOIFLAG;
             MANAME := 'A';
             CNUM := 3;
           END;
      'B': BEGIN
             CFLAG := IOBFLAG;
             MANAME := 'B';
             CNUM := 4;
           END;
    END;
    CLEARIT(1);
    IF NOT CFLAG THEN
      BEGIN
        GOTOXY(0,3);
        WRITE('THERE IS NO MATRIX ',MANAME,' YET ');
        CRT(ERASEOL);
        FOR I:=1 TO 100 DO CRT(TIME);
        SPACEBAR;
      END
    ELSE
      HARDCOPY(CNUM);
  END; (* HCOPY *)

BEGIN (* DISPLAY *)
  DISQUIT := FALSE;
  REPEAT
    CLEARSCREEN;
    HEAD('MATRIX DISPLAY',0);
    PMEN(2,'A','SCREEN DISPLAY OF A');
    PMEN(4,'B','SCREEN DISPLAY OF B');
    PMEN(6,'C','HARDCOPY OUTPUT OF A');
    PMEN(8,'D','HARDCOPY OUTPUT OF B');
    PMEN(10,'Q','QUIT');
```

```
      GOTOXY(2,12);
      WRITE('SELECT ONE : ');
      OKSET := ['A'..'D','Q'];
      DISCHOICE := GETCHAR(OKSET);
      CASE DISCHOICE OF
        'A': DDISPLA('A');
        'B': DDISPLA('B');
        'C': HCOPY('A');
        'D': HCOPY('B');
        'Q': DISQUIT := TRUE;
      END;
    UNTIL DISQUIT;
  END; (* DISPLAY *)

BEGIN (* MATRIXIO *)
  CLEARSCREEN;
  QUITFLAG := FALSE;
  REPEAT
    CLEARSCREEN;
    HEAD('INPUT MATRIX I/O',0);
    PMEN(2,'A','EDIT MATRICES');
    PMEN(4,'B','SCREEN OR HARDCOPY OUTPUT');
    PMEN(6,'Q','QUIT');
    GOTOXY(2,8);
    WRITE('SELECT ONE : ');
    OKSET := ['A','B','Q'];
    CHOICE := GETCHAR(OKSET);
    CASE CHOICE OF
      'A': EDIT;
      'B': DISPLAY;
      'Q': QUITFLAG := TRUE;
    END;
  UNTIL QUITFLAG;
END; (* MATRIXIO *)


PROCEDURE HARDCOPY(HNUM:INTEGER);
VAR F: TEXT;
    CH: CHAR;
    PRFLAG: BOOLEAN;
    RESULT: INTEGER;

  PROCEDURE SETPRINT;
  VAR CH: CHAR;
      ETRIES: INTEGER;

  BEGIN (* SETPRINT *)
    PRFLAG := FALSE;
    ETRIES := 0;
    REPEAT
      ASSIGN(F,'LST:');
      REWRITE(F);
      IF IORESULT=255 THEN
        BEGIN
          ETRIES := ETRIES+1;
```

```
        IF FTRIES<=2 THEN
          BEGIN
            WRITELN('PUT PRINTER ON LINE ');
            SPACEBAR;
          END;
      END
    ELSE
      PRFLAG := TRUE;
  UNTIL PRFLAG OR (FTRIES>2);
END; (* GETPRINT *)


PROCEDURE PRSAXB;
VAR I,J: INTEGER;
BEGIN (* PRSAXB *)
  FOR J:=1 TO IOM DO
    BEGIN
      WRITELN(F);
      WRITELN(F,'THE SOLUTION FOR COLUMN ',J,' IS  ');
      FOR I:=1 TO IONDIM DO
        BEGIN
          WRITELN(F,'[',XX[I,J],']');
        END;
    END;
END; (* PRSAXB *)

PROCEDURE PRMATRIX(AORB:CHAR);
VAR DIM,I,J: INTEGER;
    MANAME: STRING[3];
BEGIN(* PRMATRIX *)
  CASE AORB OF
    'A': BEGIN
           MANAME := 'A';
           DIM := IONDIM;
         END;
    'B': BEGIN
           MANAME := 'B';
           DIM := IOM;
         END;
  END;
  WRITELN(F);
  WRITELN(F,'THE COLUMNS OF MATRIX ',MANAME,' ARE: ');
  WRITELN(F);
  FOR J:=1 TO DIM DO
    BEGIN
      WRITELN(F);
      WRITELN(F,'COLUMN ',J,' OF ',MANAME,' IS :');
      CASE AORB OF
        'A': FOR I:=1 TO IONDIM DO
               WRITELN(F,'A[',I,',',J,']=',IOTA[I,J]);
        'B': FOR I:=1 TO IONDIM DO
               WRITELN(F,'B[',I,',',J,']=',IOTB[I,J]);
      END;
    END;
END; (* PRMATRIX *)
```

```
  PROCEDURE PRRNAA;
  VAR I,J: INTEGER;
  BEGIN (* PPRNAA *)
    WRITELN(F);
    WRITELN(F,'THE ARRAY OF COMPLEX EIGENVALUES IS');
    WRITELN(F);
    FOR I:=1 TO ICNDIM DO
      WRITELN(F,'[',TEVR[I],',',TEVI[I],']');
    WRITELN(F);
    WRITELN(F,'THE COMPLEX EIGENVECTORS ARE ');
    WRITELN(F);
    J := 1;
    REPEAT
      IF TEVI[J]<>0.0 THEN
        BEGIN
          WRITELN(F);
          WRITELN(F,'VECTOR ',J,' IS ');
          FOR I:=1 TO ICNDIM DO
            BEGIN
              WRITE(F,'VECTOR',J,'[',I,']=[',TVEC[I,J]);
              WRITELN(F,',',TVEC[I,J+1],']');
            END;
          WRITELN(F);
          WRITELN(F,'VECTOR ',J+1,' HAS COMPONENTS ');
          FOR I:=1 TO ICNDIM DO
            BEGIN
              WRITE(F,'VECTOR',J+1,'[',I,']=[',TVEC[I,J]);
              WRITELN(F,',',-TVEC[I,J+1],']');
            END;
          J := J+2;
        END
      ELSE
        BEGIN
          WRITELN(F);
          WRITELN(F,'VECTOR ',J,' HAS COMPONENTS ');
          FOR I:=1 TO ICNDIM DO
            BEGIN
              WRITE(F,'VECTOR',J,'[',I,']=[',TVEC[I,J]);
              WRITELN(F,',',0.0,']');
            END;
          J := J+1;
        END;
    UNTIL J>ICNDIM;
  END; (* PPRNAA *)

  PROCEDURE PRRDET;
  BEGIN (* PRRDET *)
    WRITELN(F);
    WRITELN(F,'THE DETERMINANT OF MATRIX A IS ',GSDET);
  END; (* PRRDERT *)

BEGIN (* HARDCOPY *)
  SETPRINT;
  IF PRFLAG THEN
```

```
    BEGIN
      CASE HNUM OF
        1: PRRNAA;
        2: PRSAXB;
        3: PRMATRIX('A');
        4: PRMATRIX('B');
        5: PRRDET;
      END;
      CLOSE(F,RESULT);
    END;
END; (* HARDCOPY *)

MODEND.
```

```
(* VERSION 0285 *)

MODULE OVERLAY5;

(* EIGENHQR MODULE *)

TYPE DOMAIN1 = 1..20;
     MATRIX = ARRAY[DOMAIN1,DOMAIN1] OF REAL;
     LISTI = ARRAY[DOMAIN1] OF INTEGER;
     LISTR = ARRAY[DOMAIN1] OF REAL;

     EXTERNAL PROCEDURE SPACEBAR;


PROCEDURE HQR2(NM,N,LOW,IGH:INTEGER;VAR H:MATRIX;VAR WR,WI:LISTR;
               VAR Z:MATRIX;VAR IERR:INTEGER);

CONST MACHEP = 1.0E-14;

VAR EN,ENM2,I,II,ITS,J,K,L,M,MP2,NA: INTEGER;
    NORM,P,Q,R,RA,S,SA,T,TEMP1,TEMP2,VR,VI,W,X,Y,ZR,ZI,ZZ: REAL;
    FLAG1,FLAG2,FLAG3,FLAG4,NOTLAS,TESTEXIT: BOOLEAN;
    CH: CHAR;

  FUNCTION SIGN(E,F:REAL): REAL;
  BEGIN (* SIGN *)
    IF F<0 THEN
      SIGN := -ABS(E)
    ELSE
      SIGN := ABS(E);
  END; (* SIGN *)

  FUNCTION MIN0(I1,I2:INTEGER): INTEGER;
  BEGIN (* MIN0 *)
    IF I1<I2 THEN
      MIN0 := I1
    ELSE
      MIN0 := I2;
  END; (* MIN0 *)

  PROCEDURE COMDIV(XR,XI,YR,YI:REAL;VAR ZR,ZI:REAL);
  VAR D,H: REAL;
  BEGIN (* COMDIV *)
    IF ABS(YR)<ABS(YI) THEN
      BEGIN
        H := YR/YI;
        D := YI+H*YR;
        ZR := (XR*H+XI)/D;
        ZI := (XI*H-XR)/D;
      END
    ELSE
      BEGIN
        H := YI/YR;
        D := YR+H*YI;
```

```
      ZR := (XR+H*XI)/D;
      ZI := (XI-H*XR)/D;
    END;
END; (* COMDIV *)


PROCEDURE HQROPT1;
VAR I: INTEGER;

  PROCEDURE LOOK;
  VAR I: INTEGER;
  BEGIN(* LOOK *)
    M := ENM2+1;
    FLAG2 := FALSE;
    REPEAT
      M := M-1;
      ZZ := H[M,M];
      R := X-ZZ;
      S := Y-ZZ;
      P := (R*S-W)/H[M+1,M]+H[M,M+1];
      Q := H[M+1,M+1]-ZZ-R-S;
      R := H[M+2,M+1];
      S := ABS(P)+ABS(Q)+ABS(R);
      P := P/S;
      Q := Q/S;
      R := R/S;
      IF M=L THEN
        FLAG2 := TRUE
      ELSE
        BEGIN
          TEMP1 := ABS(H[M,M-1])*(ABS(Q)+ABS(R));
          TEMP2 := ABS(H[M-1,M-1])+ABS(ZZ)+ABS(H[M+1,M+1]);
          TEMP2 := MACHEP*ABS(P)*TEMP2;
          IF TEMP1 <= TEMP2 THEN
            FLAG2 := TRUE;
        END;
      WRITE('.');
    UNTIL FLAG2=TRUE;
    MP2 := M+2;
    FOR I:=MP2 TO EN DO
      BEGIN
        H[I,I-2] := 0.0;
        IF I<>MP2 THEN
          H[I,I-3] := 0.0;
      END;
  END;(* LOOK *)


  PROCEDURE DOUBLEQR;
  VAR M: INTEGER;

    PROCEDURE COLMOD;
    VAR I: INTEGER;
```

```
  BEGIN (* COLMOD *)
    FOR I:=1 TO J DO
      BEGIN
        P := X*H[I,K]+Y*H[I,K+1];
        IF NOTLAS THEN
          BEGIN
            P := P+ZZ*H[I,K+2];
            H[I,K+2] := H[I,K+2]-P*R;
          END;
        H[I,K+1] := H[I,K+1]-P*Q;
        H[I,K] := H[I,K]-P;
      END;
  END; (* COLMOD *)

  PROCEDURE ROWMOD;
  VAR J: INTEGER;
  BEGIN (* ROWMOD *)
    FOR J:=K TO N DO
      BEGIN
        P := H[K,J]+Q*H[K+1,J];
        IF NOTLAS THEN
          BEGIN
            P := P+R*H[K+2,J];
            H[K+2,J] := H[K+2,J]-P*ZZ;
          END;
        H[K+1,J] := H[K+1,J]-P*Y;
        H[K,J] := H[K,J]-P*X;
      END;
  END; (* ROWMOD *)


  PROCEDURE ACCTRANS;
  VAR I: INTEGER;
  BEGIN(* ACCTRANS *)
    FOR I:=LOW TO IGH DO
      BEGIN
        P := X*Z[I,K]+Y*Z[I,K+1];
        IF NOTLAS THEN
          BEGIN
            P := P+ZZ*Z[I,K+2];
            Z[I,K+2] := Z[I,K+2]-P*R;
          END;
        Z[I,K+1] := Z[I,K+1]-P*Q;
        Z[I,K] := Z[I,K]-P;
      END;
  END;(* ACCTRANS *)

BEGIN(* DOUBLEQR *)
  FOR K:=M TO NA DO
    BEGIN
      FLAG1 := FALSE;
      NOTLAS := K<>NA;
      IF K<>M THEN
        BEGIN
          P := H[K,K-1];
```

```
            Q := H[K+1,K-1];
            R := 0.0;
            IF NOTLAS THEN
              R := H[K+2,K-1];
            X := ABS(P)+ABS(Q)+ABS(R);
            IF X=0.0 THEN
              FLAG1 := TRUE
            ELSE
              BEGIN
                P := P/X;
                Q := Q/X;
                R := R/X;
              END;
          END;
        IF FLAG1=FALSE THEN
          BEGIN
            S := P*P+Q*Q+R*R;
            S := SQRT(S);
            S := SIGN(S,P);
            IF K=1 THEN
              BEGIN
                IF L<>M THEN
                  H[K,K-1] := -H[K,K-1]
              END
            ELSE
              H[K,K-1] := -S*X;
            P := P+S;
            X := P/S;
            Y := Q/S;
            ZZ := R/S;
            Q := Q/P;
            R := R/P;
            ROWMOD;
            WRITE('.');
            J := MIN0(EN,K+3);
            COLMOD;
            WRITE('.');
            ACCTRANS;
            WRITE('.');
          END;
      END;
  END;(* DOUBLEQR *)

BEGIN (* HQRCPT1 *)
  TESTEXIT := FALSE;
  IF ITS>=100 THEN
    BEGIN
      IERR := EN;
      WRITELN('THE ITERATION LIMIT OF 100 HAS BEEN ');
      WRITELN('   REACHED IN -HQR-');
      SPACEBAR;
      TESTEXIT := TRUE;
    END
  ELSE
    BEGIN
```

```
      IF (ITS<>0) AND (ITS MOD 10=0) THEN
        BEGIN
          T := T+X;
          FOR I:=LOW TO EN DO
            H[I,I] := H[I,I]-X;
          S := ABS(H[EN,NA])+ABS(H[NA,EN-1]);
          X := 0.75*S;
          Y := X;
          W := -0.4375*S*S;
        END;
      ITS := ITS+1;
      LOOK;
      DOUBLEQR;
    END;
END;(* HQROPT1 *)

PROCEDURE HQROPT2;

  PROCEDURE ROWMOD2;
  VAR J: INTEGER;

  BEGIN(* ROWMOD2 *)
    FOR J:=NA TO N DO
      BEGIN
        ZZ := H[NA,J];
        H[NA,J] := Q*ZZ+P*H[EN,J];
        H[EN,J] := Q*H[EN,J]-P*ZZ;
      END;
  END;(* ROWMOD2 *)


  PROCEDURE COLMOD2;
  VAR I: INTEGER;
  BEGIN(* COLMOD2 *)
    FOR I:=1 TO EN DO
      BEGIN
        ZZ := H[I,NA];
        H[I,NA] := Q*ZZ+P*H[I,EN];
        H[I,EN] := Q*H[I,EN]-P*ZZ;
      END;
  END;(* COLMOD2 *)

  PROCEDURE ACCTRANS2;
  VAR I: INTEGER;
  BEGIN (* ACCTRANS2 *)
    FOR I:=LOW TO IGH DO
      BEGIN
        ZZ := Z[I,NA];
        Z[I,NA] := Q*ZZ+P*Z[I,EN];
        Z[I,EN] := Q*Z[I,EN]-P*ZZ;
      END;
  END;(* ACCTRANS2 *)

BEGIN (* HQROPT2 *)
  P := (Y-X)/2.0;
```

```
    Q := P*P+W;
    ZZ := SQRT(ABS(Q));
    H[EN,EN] := X+T;
    X := H[EN,EN];
    H[NA,NA] := Y+T;
    IF Q<0.0 THEN
      BEGIN
        WR[NA] := X+P;
        WR[EN] := X+P;
        WI[NA] := ZZ;
        WI[EN] := -ZZ;
      END
    ELSE
      BEGIN
        ZZ := P+SIGN(ZZ,P);
        WR[NA] := X+ZZ;
        WR[EN] := WR[NA];
        IF ZZ<>0.0 THEN
          WR[EN] := X-W/ZZ;
        WI[NA] := 0.0;
        WI[EN] := 0.0;
        X := H[EN,NA];
        R := SQRT(X*X+ZZ*ZZ);
        P := X/R;
        Q := ZZ/R;
        ROWMOD2;
        WRITE('.');
        COLMOD2;
        WRITE('.');
        ACCTRANS2;
        WRITE('.');
      END;
    EN := ENM2;
    IF EN>= LOW THEN
      BEGIN
        ITS := 0;
        NA := EN-1;
        ENM2 := NA-1;
      END
    ELSE
      FLAG3 := TRUE;
  END;(* HQROPT2 *)

  PROCEDURE EIGENVECTOR;
  VAR EN,I,J: INTEGER;

    PROCEDURE LOOP700;
    VAR I,II,J: INTEGER;
        TTP: REAL;
    BEGIN(* LOOP700 *)
      WRITE('.');
      FOR II:=1 TO NA DO
        BEGIN
          I := EN-II;
          W := H[I,I]-P;
```

```
      R := H[I,EN];
      IF M<=NA THEN
        FOR J:=M TO NA DO
          BEGIN
            IF (H[I,J]=0.0) OR (H[J,EN]=0.0) THEN
              TTP := 0.0
            ELSE
              TTP := H[I,J]*H[J,EN];
            R := R+TTP;
          END;
      IF WI[I]<0.0 THEN
        BEGIN
          ZZ := W;
          S := R;
        END
      ELSE
        BEGIN
          M := I;
          IF WI[I]=0.0 THEN
            BEGIN
              T := W;
              IF W=0.0 THEN
                T := MACHEP*NORM1;
              H[I,EN] := -R/T;
            END
          ELSE
            BEGIN
              X := H[I,I+1];
              Y := H[I+1,I];
              Q := (WR[I]-P)*(WR[I]-P)+WI[I]*WI[I];
              T := (X*S-ZZ*R)/Q;
              H[I,EN] := T;
              IF ABS(X)>ABS(ZZ) THEN
                H[I+1,EN] := (-R-W*T)/X
              ELSE
                H[I+1,EN] := (-S-Y*T)/ZZ;
            END;
        END;
    END;
END;(* LOOP700 *)


PROCEDURE LOOP790;
VAR II,J: INTEGER;

  PROCEDURE LOOPWORK;
  BEGIN (* LOOPWORK *)
    X := H[I,I+1];
    Y := H[I+1,I];
    VR := (WR[I]-P)*(WR[I]-P)+WI[I]*WI[I]-Q*Q;
    VI := (WR[I]-P)*2.0*Q;
    IF (VR=0.0) AND (VI=0.0) THEN
      VR := MACHEP*NORM*(ABS(W)+ABS(Q)+
            ABS(X)+ABS(Y)+ABS(ZZ));
    COMDIV(X*R-ZZ*RA+Q*SA,X*S-ZZ*SA-Q*RA,
```

```
            VR,VI,ZR,ZI);
      H[I,NA] := ZR;
      H[I,EN] := ZI;
      IF ABS(X)>(ABS(ZZ)+ABS(Q)) THEN
        BEGIN
          H[I+1,NA] := (-RA-W*H[I,NA]+Q*H[I,EN])/X;
          H[I+1,EN] := (-SA-W*H[I,EN]-Q*H[I,NA])/X;
        END
      ELSE
        BEGIN
          COMDIV(-R-Y*H[I,NA],-S-Y*H[I,EN],ZZ,Q,ZR,ZI);
          H[I+1,NA] := ZR;
          H[I+1,EN] := ZI;
        END;
    END; (* LOOPWORK *)

BEGIN(* LOOP790 *)
  FOR II:=1 TO ENM2 DO
    BEGIN
      WRITE('.');
      I := NA-II;
      W := H[I,I]-P;
      RA := 0.0;
      SA := H[I,EN];
      FOR J:=M TO NA DO
        BEGIN
          RA := RA+H[I,J]*H[J,NA];
          SA := SA+H[I,J]*H[J,EN];
        END;
      IF WI[I]<0.0 THEN
        BEGIN
          ZZ := W;
          R := RA;
          S := SA;
        END
      ELSE
        BEGIN
          M := I;
          IF WI[I]=0.0 THEN
            BEGIN
              COMDIV(-RA,-SA,W,Q,ZR,ZI);
              H[I,NA] := ZR;
              H[I,EN] := ZI;
            END
          ELSE
            LOOPWORK;
        END;
    END;
END;(* LOOP790 *)



PROCEDURE LOOP830;
VAR I,J,K: INTEGER;
```

```
  BEGIN(* LOOP880 *)
    FOR J:=N DOWNTO LOW DO
      BEGIN
        WRITE('.');
        M := MIN0(J,IGH);
        FOR I:=LOW TO IGH DO
          BEGIN
            ZZ := 0.0;
            FOR K:=LOW TO M DO
              ZZ := ZZ+Z[I,K]*H[K,J];
            Z[I,J] := ZZ;
          END;
      END;
  END;(* LOOP880 *)

  PROCEDURE PRELOOP;
  BEGIN (* PRELOOP *)
    I := NA;
    IF ABS(H[EN,NA])>ABS(H[NA,EN]) THEN
      BEGIN
        H[NA,NA] := Q/H[EN,NA];
        H[NA,EN] := -(H[EN,EN]-P)/H[EN,NA];
      END
    ELSE
      BEGIN
        COMDIV(0.0,-H[NA,EN],H[NA,NA]-P,Q,ZR,ZI);
        H[NA,NA] := ZR;
        H[NA,EN] := ZI;
      END;
    H[EN,NA] := 0.0;
    H[EN,EN] := 1.0;
    ENM2 := NA-1;
  END; (* PRELOOP *)

BEGIN(* EIGENVECTOR *)
  NORM := 0.0;
  K := 1;
  FOR I:=1 TO N DO
    BEGIN
      FOR J:=K TO N DO
        NORM := NORM+ABS(H[I,J]);
      K := I;
    END;
  IF NORM<>0.0 THEN
    BEGIN
      FOR EN:=N DOWNTO 1 DO
        BEGIN
          P := WR[EN];
          Q := WI[EN];
          NA := EN-1;
          IF Q<=0.0 THEN
            BEGIN
              IF Q=0.0 THEN
                BEGIN
                  M := EN;
```

```
                    H[EN,EN] := 1.0;
                    IF NA<>0 THEN
                      LOOP700;
                  END
                ELSE
                  BEGIN
                    PRELOOP;
                    IF ENM2<>0 THEN
                      LOOP790;
                  END;
              END;
          END;
        FOR I:=1 TO N DO
          IF (I<LOW) OR (I>IGH) THEN
            FOR J:=I TO N DO
              Z[I,J] := H[I,J];
        LOOP880;
      END;
  END;(* EIGENVECTOR *)

BEGIN(* HQR2 *)
  IERR := 0;
  FOR I:=1 TO N DO
    BEGIN
      IF (I<LOW) OR (I>IGH) THEN
        BEGIN
          WR[I] := H[I,I];
          WI[I] := 0.0;
        END;
    END;
  EN := IGH;
  T := 0.0;
  IF EN>=LOW THEN
    BEGIN
      ITS := 0;
      NA := EN-1;
      ENM2 := NA-1;
      FLAG3 := FALSE;
      REPEAT
        FLAG4 := FALSE;
        L := EN+1;
        REPEAT
          L := L-1;
          IF L=LOW THEN
            FLAG4 := TRUE
          ELSE
            IF ABS(H[L,L-1])<=MACHEP*(ABS(H[L-1,L-1])+ABS(H[L,L])
               ) THEN
              FLAG4 := TRUE;
        UNTIL FLAG4=TRUE;
        X := H[EN,EN];
        IF L<>EN THEN
          BEGIN
            Y := H[NA,NA];
            W := H[EN,NA]*H[NA,EN];
```

```
            IF L<>NA THEN
              BEGIN
                HQROPT1;
                IF TESTEXIT THEN
                  EXIT;
              END
            ELSE
              HQROPT2;
          END
        ELSE
          BEGIN
            H[EN,EN] := X+T;
            WR[EN] := H[EN,EN];
            WI[EN] := 0.0;
            EN := NA;
            IF EN>=LOW THEN
              BEGIN
                ITS := 0;
                NA := EN-1;
                ENM2 := NA-1;
              END
            ELSE
              FLAG3 := TRUE;
          END;
      UNTIL FLAG3;
      EIGENVECTOR;
    END;
END;(* HQR2 *)

.CODEND.
```

```
(* VERSION 0286 *)

MODULE OVERLAY6;
(* MODULE EIGENBAL *)

TYPE DOMAIN1 = 1..20;
     MATRIX = ARRAY[DOMAIN1,DOMAIN1] OF REAL;
     LISTI = ARRAY[DOMAIN1] OF INTEGER;
     LISTR = ARRAY[DOMAIN1] OF REAL;

PROCEDURE ELMHES(NM,N,LOW,IGH:INTEGER;VAR A:MATRIX;VAR INT:LISTI);
VAR I,J,KP1,LA,M,MM1,MP1: INTEGER;
    X,Y: REAL;

BEGIN (* ELMHES *)
  LA := IGH-1;
  KP1 := LOW+1;
  IF LA>=KP1 THEN
    BEGIN
      FOR M:=KP1 TO LA DO
        BEGIN
          MM1 := M-1;
          X := 0.0;
          I := M;
          FOR J:=M TO IGH DO
            BEGIN
              IF ABS(A[J,MM1])>ABS(X) THEN
                BEGIN
                  WRITE('.');
                  X := A[J,MM1];
                  I := J;
                END;
            END;
          INT[M] := I;
          IF I<>M THEN
            BEGIN
              FOR J:=MM1 TO N DO
                BEGIN
                  Y := A[I,J];
                  A[I,J] := A[M,J];
                  A[M,J] := Y;
                END;
              FOR J:=1 TO IGH DO
                BEGIN
                  Y := A[J,I];
                  A[J,I] := A[J,M];
                  A[J,M] := Y;
                END;
            END;
          IF X<>0.0 THEN
            BEGIN
              MP1 := M+1;
              FOR I:=MP1 TO IGH DO
                BEGIN
                  Y := A[I,MM1];
```

```
                    IF Y<>0.0 THEN
                      BEGIN
                        WRITE('.');
                        Y := Y/X;
                        A[I,MM1] := Y;
                        FOR J:=M TO N DO
                          A[I,J] := A[I,J]-Y*A[M,J];
                        FOR J:=1 TO IGH DO
                          A[J,M] := A[J,M]+Y*A[J,I];
                      END;
                  END;
              END;
          END;
      END;
END; (* ELMHES *)


PROCEDURE ELTRAN(NM,N,LOW,IGH:INTEGER;A:MATRIX;
                 JN:LISTI;VAR Z:MATRIX);
VAR I,J,KL,MM,MP,MP1: INTEGER;

BEGIN(* ELTRAN *)
  FOR I:=1 TO N DO
    BEGIN
      FOR J:=1 TO N DO
        Z[I,J] := 0.0;
      Z[I,I] := 1.0;
    END;
  KL := IGH-LOW-1;
  IF KL>=1 THEN
    BEGIN
      FOR MM:=1 TO KL DO
        BEGIN
          MP := IGH-MM;
          MP1 := MP+1;
          FOR I:=MP1 TO IGH DO
            Z[I,MP] := A[I,MP-1];
          I := JN[MP];
          WRITE('.');
          IF I<>MP THEN
            BEGIN
              FOR J:=MP TO IGH DO
                BEGIN
                  Z[MP,J] := Z[I,J];
                  Z[I,J] := 0.0;
                END;
              Z[I,MP] := 1.0;
            END;
        END;
    END;
END;(* ELTRAN *)

PROCEDURE BALBAK(NM,N,LOW,IGH:INTEGER;SCALE:LISTR;
                 M:INTEGER;VAR Z:MATRIX);
```

```
VAR I,J,K,II: INTEGER;
    S: REAL;

BEGIN(* BALBAK *)
  IF IGH<>LOW THEN
    BEGIN
      FOR I:=LOW TO IGH DO
        BEGIN
          S := SCALE[I];
          FOR J:=1 TO M DO
            Z[I,J] := Z[I,J]*S;
        END;
    END;
  FOR II:=1 TO N DO
    BEGIN
      I := II;
      IF (I<LOW) OR (I>IGH) THEN
        BEGIN
          IF I<LOW THEN
            I := LOW-II;
          K := TRUNC(SCALE[I]);
          IF K<>I THEN
            BEGIN
              WRITE('.');
              FOR J:=1 TO M DO
                BEGIN
                  S := Z[I,J];
                  Z[I,J] := Z[K,J];
                  Z[K,J] := S;
                END;
            END;
        END;
    END;
END;(* BALBAK *)

PROCEDURE BALANCE(NM1,N:INTEGER;VAR A:MATRIX;VAR LOW,IGH:INTEGER;
                  VAR SCALE:LISTR);
CONST RADIX = 2.0;

VAR FLAGA,FLAGB,FLAGC,FLAGD,FLAG1,FLAG2,NOCONV: BOOLEAN;
    I,IEXC,J,K,L,M: INTEGER;
    B2,C,F,G,R,S: REAL;

  PROCEDURE ROWSEARCH;
  BEGIN (* ROWSEARCH *)
    J := L+1;
    FLAGA := TRUE;
    REPEAT
      J := J-1;
      I := 0;
      FLAG2 := FALSE;
      REPEAT
        I := I+1;
        IF I<>J THEN
          BEGIN
```

```
          IF A[J,I]<>0.0 THEN
            FLAG2 := TRUE;
        END;
    UNTIL (I=L) OR (FLAG2=TRUE);
    IF NOT FLAG2 THEN
      BEGIN
        I := L;
        IEXC := 1;
        FLAGA := FALSE;
      END;
  UNTIL (J=1) OR (FLAGA=FALSE);
END; (* ROWSEARCH *)


PROCEDURE COLUMNSEARCH;
BEGIN (* COLUMNSEARCH *)
  J := K-1;
  FLAGB := TRUE;
  REPEAT
    J := J+1;
    I := K-1;
    FLAG1 := FALSE;
    REPEAT
      I := I+1;
      IF I<>J THEN
        BEGIN
          IF A[I,J]<>0.0 THEN
            FLAG1 := TRUE;
        END;
    UNTIL (I=L) OR (FLAG1=TRUE);
    IF NOT FLAG1 THEN
      BEGIN
        I := K;
        IEXC := 2;
        FLAGB := FALSE;
      END;
  UNTIL (J=L) OR (FLAGB=FALSE);
END; (* COLUMNSEARCH *)


PROCEDURE NORMREDUCTION;
VAR I,J: INTEGER;
BEGIN  (* NORMREDUCTION *)
  FOR I:=K TO L DO
    BEGIN
      C := 0.0;
      R := 0.0;
      FOR J:=K TO L DO
        BEGIN
          IF J<>I THEN
            BEGIN
              C := C+ABS(A[J,I]);
              R := R+ABS(A[I,J]);
            END;
        END;
```

```
          G := R/RADIX;
          WRITE('.');
          F := 1.0;
          S := C+R;
          WHILE C<G DO
            BEGIN
              F := F*RADIX;
              C := C*B2;
            END;
          G := R*RADIX;
          WHILE C>=G DO
            BEGIN
              F := F/RADIX;
              C := C/B2;
            END;
          IF ((C+R)/F)<(0.95*S) THEN
            BEGIN
              G := 1.0/F;
              SCALE[I] := SCALE[I]*F;
              NOCONV := TRUE;
              FOR J:=K TO N DO
                A[I,J] := A[I,J]*G;
              FOR J:=1 TO L DO
                A[J,I] := A[J,I]*F;
            END;
        END;
    END; (* NORMREDUCTION *)

BEGIN (* BALANCE *)
    B2 := RADIX*RADIX;
    K := 1;
    L := N;
    FLAGD := FALSE;
    WHILE NOT FLAGD DO
      BEGIN
        FLAGD := TRUE;
        FLAGB := FALSE;
        ROWSEARCH;
        IF FLAGA THEN
          COLUMNSEARCH;
        IF NOT FLAGB THEN
          BEGIN
            REPEAT
              SCALE[M] := J;
              IF J<>M THEN
                BEGIN
                  FOR I:=1 TO L DO
                    BEGIN
                      F := A[I,J];
                      A[I,J] := A[I,M];
                      A[I,M] := F;
                    END;
                  FOR I:=K TO N DO
                    BEGIN
                      F := A[J,I];
```

```
                    A[J,I] := A[M,I];
                    A[M,I] := F;
                  END;
              END;
            IF IEXC=1 THEN
              FLAGC := TRUE
            ELSE
              FLAGC := FALSE;
            IF NOT FLAGC THEN
              BEGIN
                K := K+1;
                COLUMNSEARCH;
              END;
            WRITE('.');
          UNTIL FLAGC OR FLAGB;
          IF NOT FLAGB THEN
            BEGIN
              IF L<>1 THEN
                BEGIN
                  FLAGB := FALSE;
                  L := L-1;
                END;
            END;
        END;
    END;
  IF FLAGB THEN
    BEGIN
      FOR I:=K TO L DO
        SCALE[I] := 1.0;
      REPEAT
        NOCONV := FALSE;
        NORMREDUCTION;
      UNTIL NOT NOCONV;
    END;
  LOW := K;
  IGH := L;
END; (* BALANCE *)


MODEND.
```

```
(* VERSION 0285 *)
(* MODULE DETERM *)

MODULE OVERLAY10;

TYPE DOMAIN1 = 1..20;
     MATRIX = ARRAY[DOMAIN1,DOMAIN1] OF REAL;
     LISTI = ARRAY[DOMAIN1] OF INTEGER;
     LISTR = ARRAY[DOMAIN1] OF REAL;

VAR IONDIM: EXTERNAL INTEGER;
    AOIFLAG: EXTERNAL BOOLEAN;
    IOTA: EXTERNAL MATRIX;
    GSDET: EXTERNAL REAL;

    EXTERNAL PROCEDURE CLEARSCREEN;

    EXTERNAL PROCEDURE SPACEBAR;

    EXTERNAL FUNCTION YES: BOOLEAN;

    EXTERNAL [3] PROCEDURE HARDCOPY(HNUM:INTEGER);

    EXTERNAL [1] PROCEDURE RLUD(ND,N:INTEGER;VAR KER:INTEGER;VAR AL
                                MATRIX;
                                VAR JN:LISTI; VAR SCALE:LISTR);

PROCEDURE FTRDET;
VAR TFLD: INTEGER;
    TSCALE: LISTR;
    JIN: LISTI;
    SDET: REAL;
    I,J,PDEG,NT:INTEGER;
    CHOICE,SELECTION: CHAR;
    TCOEF: LISTR;
    TNI,TNR: LISTR;
    IERR: INTEGER;
    CH: CHAR;
    SREAL: STRING;
    P1: INTEGER;
    TALU:MATRIX;

  PROCEDURE RDET(SND,SN:INTEGER;VAR SKER:INTEGER;VAR DET:REAL;
                 VAR SA:MATRIX);


  VAR J: INTEGER;
      SJN: LISTI;
      SSCALE: LISTR;
      P: STRING;
      CH: CHAR;

  BEGIN (* RDET *)
    SKER := 0;
    RLUD(SND,SN,SKER,SA,SJN,SSCALE);
```

```
    DET := SJN[SN];
    FOR J:=1 TO SN DO
      DET := DET*SA[J,J];
  END;(* RDET *)

BEGIN (* TTPDET *)
  CLEARSCREEN;
  NT := ICNDIT;
  FOR I:=1 TO NT DO
    FOR J:= 1 TO NT DO
      TALU[I,J] := ICTA[I,J];
  CLEARSCREEN;
  WRITELN('PLEASE WAIT ');
  RDET(NT,NT,TKER,SDET,TALU);
  CLEARSCREEN;
  IF TKER=0 THEN
    BEGIN
      GDET := SDET;
      WRITELN;
      WRITELN('THE DETERMINANT = ',SDET);
      SPACEBAR;
      WRITELN;
      WRITE('DO YOU WANT A HARDCOPY? Y/N ');
      IF YES THEN
        HARDCOPY(5);
    END;
END; (* TTRDET *)

MODEND.
```

```
(* VERSION 300 *)
(* AX=B MODULE *)
MODULE OVERLAY20;

TYPE DOMAIN1 = 1..20;
     MATRIX = ARRAY[DOMAIN1,DOMAIN1] OF REAL;
     LISTI = ARRAY[DOMAIN1] OF INTEGER;
     LISTR = ARRAY[DOMAIN1] OF REAL;


VAR IOTP,IOTB,XX: EXTERNAL MATRIX;
    ACIFLAG,IOBFLAG: EXTERNAL BOOLEAN;
    IONDIM,ION: EXTERNAL INTEGER;

    EXTERNAL FUNCTION YES: BOOLEAN;

    EXTERNAL PROCEDURE SPACEBAR;

    EXTERNAL PROCEDURE CLEARSCREEN;

    EXTERNAL [1] PROCEDURE BLUD(ND,N:INTEGER;VAR KER:INTEGER;V
                                MATRIX;
                                VAR JN:LISTI; VAR SCALE:LISTR);

    EXTERNAL [1] PROCEDURE REDS(ND,N:INTEGER;VAR KER:INTEGER;VAR
                                MATRIX;
                                VAR JN:LISTI; VAR X:LISTR);

    EXTERNAL [2] PROCEDURE HARDCOPY(HNUM:INTEGER);

PROCEDURE TTSAX2;

VAR QUITFLAG: BOOLEAN;
    I,J,PDES: INTEGER;
    CHOICE,SELECTION: CHAR;
    TCOEF: LISTR;
    TNI,TVR: LISTR;
    IERP,MT,NT: INTEGER;
    CH: CHAR;
    SREEL: STRING;
    P4: INTEGER;
    TKER: INTEGER;
    TSCALE: LISTR;
    TIN: LISTI;
    TINIT: INTEGER;
    TALU,TB:MATRIX;

  PROCEDURE SAXB(SND,SN,SM:INTEGER;VAR SA:MATRIX;VAR SB,SX:MATRIX;
                 VAR SINIT:INTEGER;VAR SJN:LISTI;VAR SKER:INTEGER);
  VAR I,J,NN: INTEGER;
      SB,SCALE: LISTR;
  BEGIN (* SAXB *)
    NN := SN;
    IF SINIT=0 THEN
```

```
      RLUD(SND,SN,SKER,SA,SJN,SCALE);
    IF SJN[SN]=0 THEN
      BEGIN
        SKER := 3;
        WRITELN;
        WRITELN('IN SAXB, LU DECOMPOSITION OF A ');
        WRITELN('YIELDED A SINGULAR  U . A UNIQUE ');
        WRITELN('SOLUTION DOES NOT EXIST.');
        SPACEBAR;
        EXIT;
      END;
    IF SM<=0 THEN
      BEGIN
        SKER := 0;
        EXIT;
      END;
    FOR J:=1 TO SM DO
      BEGIN
        FOR I:=1 TO NN DO
          BB[I] := SB[I,J];
        RBAS(SND,SN,SKER,SA,SJN,BB);
        FOR I:=1 TO NN DO
          SX[I,J] := BB[I];
      END;
    WRITELN;
  END; (* SAXB *)

BEGIN(* TTSAXB *)
  CLEARSCREEN;
  NT:= ICNDIM;
  MT:= ICM;
  TINIT := 0;
  FOR I:=1 TO NT DO
    FOR J:= 1 TO NT DO
      TALU[I,J] := ICTA[I,J];
  FOR J:=1 TO MT DO
    FOR I:=1 TO NT DO
      TB[I,J] := ICTB[I,J];
  CLEARSCREEN;
  WRITELN('PLEASE WAIT ');
  SAXB(NT,NT,MT,TALU,TB,XX,TINIT,TIN,TKER);
  CLEARSCREEN;
  IF TKER=0 THEN
    BEGIN
      FOR J:=1 TO  MT DO
        BEGIN
          WRITELN('THE SOLUTION FOR COLUMN ',J,' IS ');
          FOR I:=1 TO NT DO
            BEGIN
              WRITE('[');
              WRITE(XX[I,J]);
              WRITELN(']');
            END;
          SPACEBAR;
          WRITELN;
```

```
        END;
        WRITELN;
        WRITE('DO YOU WANT A HARDCOPY? Y/N ');
        IF YES THEN
             HARDCOPY(2);
    END;
END; (* TTSAX3 *)

ICDEND.
```

```
(* VERSION 1283 *)
(* POLYROOT MODULE *)

MODULE OVERLAY21;

CONST MAXDEGP1 = 21;
      MAXDEG = 20;


TYPE DOMAIN1 = 1..20;
     DOMRP1 = 1..MAXDEGP1;
     MATRIX = ARRAY[DOMAIN1,DOMAIN1] OF REAL;
     LISTI = ARRAY[DOMAIN1] OF INTEGER;
     LISTR = ARRAY[DOMAIN1] OF REAL;
     LISTRP1 = ARRAY[DOMRP1] OF REAL;

     EXTERNAL PROCEDURE CLEARSCREEN;

     EXTERNAL PROCEDURE CLEARBIT(I:INTEGER);

     EXTERNAL PROCEDURE INTREAD(VAR K:INTEGER);

     EXTERNAL PROCEDURE SPACEBAR;


     EXTERNAL [3] PROCEDURE GETPOLCOF(NROOT:INTEGER;VAR PCOEF:LISTRP1);


     EXTERNAL [5] PROCEDURE HQR2(NM1,N,LOW,IGH:INTEGER; VAR H:MATRIX; V
                                 WR,WI:
                                 LISTR; VAR Z:MATRIX; VAR IERR:INTEGER

     EXTERNAL [6] PROCEDURE BALANCE(NM1,N:INTEGER; VAR A:MATRIX; VAR LO
                                    IGH:
                                    INTEGER; VAR SCALE: LISTR);

     EXTERNAL [6] PROCEDURE ELMHES(NM1,N,LOW,IGH:INTEGER; VAR A:MATRIX;
                                   VAR INT:
                                   LISTI);

     EXTERNAL [6] PROCEDURE ELTRAN(NM1,N,LOW,IGH:INTEGER; VAR A:MATRIX;
                                   INT: LISTI;
                                   VAR Z: MATRIX);

PROCEDURE TTRPQR;
VAR QUITFLAG: BOOLEAN;
    I,J,PDEG: INTEGER;
    CHOICE,SELECTION: CHAR;
    PROOEF: LISTRP1;
    TWI,TWR: LISTR;
    TIERR,NDEG: INTEGER;
    CH: CHAR;
    SREEL: STRING;
    P4: INTEGER;
    TABSERR,TRELERR: LISTR;
```

```
  TKLUST: LISTI;
  N1,TKER: INTEGER;


PROCEDURE REND2(RN:INTEGER;COEF:LISTRP1;RWR,RWI:LISTR;
                VAR ABSERR,RELERR:LISTR;VAR KLUST:LISTI;
                VAR KER:INTEGER);

VAR I,J,JP1,JF,K,M,NM1,NR1: INTEGER;
    A,CERT,DIST,EMAG,MAG,OLDERR,P,POWER,R,RAT,
    SVR,SVI,UNCERT,VR,VI,VT,XMAG,XI,XR: REAL;
    SHRUNK: BOOLEAN;

  FUNCTION CPABS(XR,YR:REAL): REAL;
  BEGIN (* CPABS *)
    CPABS := SQRT(XR*XR+YR*YR);
  END; (* CPABS *)


  FUNCTION AMAX1(A,B:REAL): REAL;
  BEGIN (* AMAX1 *)
    IF A<B THEN
      AMAX1 := B
    ELSE
      AMAX1 := A;
  END; (* AMAX1 *)

  FUNCTION SINGLE(VALUE:REAL): REAL;
  BEGIN (* SINGLE *)
    SINGLE := VALUE;
  END; (* SINGLE *)

  FUNCTION DOUBLE(VALUE:REAL): REAL;
  BEGIN (* DOUBLE *)
    DOUBLE := VALUE;
  END; (* DOUBLE *)

  PROCEDURE DECHALF;
  VAR J,K: INTEGER;
  BEGIN (* DECHALF *)
    FOR J:=1 TO RN DO
      BEGIN
        WRITE('...');
        KLUST[J] := 1;
        XMAG := CPABS(RWR[J],RWI[J]);
        EMAG := ABSERR[J];
        IF EMAG=0.0 THEN
          R := 0.0
        ELSE
          BEGIN
            IF XMAG=0.0 THEN
              R := -1.0
            ELSE
              R := EMAG/XMAG;
```

```
          END;
        RELERR[J] := R;
      END;
    NM1 := RN-1;
    FOR J:=1 TO NM1 DO
      BEGIN
        WRITE('...');
        JP1 := J+1;
        FOR K:=JP1 TO RN DO
          BEGIN
            DIST := CPABS(RWR[J]-RWR[K],RWI[J]-RWI[K]);
            IF DIST<=(ABSERR[J]+ABSERR[K]) THEN
              BEGIN
                KLUST[J] := KLUST[J]+1;
                KLUST[K] := KLUST[K]+1;
              END;
          END;
      END;
    KER := 0;
  END; (* SECHALF *)


BEGIN (* RBND2 *)
  WRITELN;
  IF RN<1 THEN
    BEGIN
      KER := 2;
      WRITELN('N(DEGREE) MUST BE >= 1');
      SPACEBAR;
      EXIT;
    END;
  NP1 := RN+1;
  POWER := 1.0/RN;
  P := ABS(COEF[1]);
  IF P=0.0 THEN
    BEGIN
      KER := 2;
      WRITELN('LEADING COEFFICIENT IS ZERO.');
      SPACEBAR;
      EXIT;
    END;
  RAT := ABS(COEF[NP1])/P;
  RAT := RAT*EXP((-45.0)*LN(2.0));
  FOR JR:=1 TO RN DO
    BEGIN
      WRITE('...');
      XR := DOUBLE(RWR[JR]);
      XI := DOUBLE(RWI[JR]);
      VR := DOUBLE(0.0);
      VI := DOUBLE(0.0);
      FOR J:=1 TO NP1 DO
        BEGIN
          WRITE('...');
          VT := XR*VR-XI*VI+DOUBLE(COEF[J]);
          VI := XR*VI+XI*VR;
```

```
          VR := VT;
        END;
      SVR := SINGLE(VR);
      SVI := SINGLE(VI);
      MAG := CPABS(SVR,SVI);
      B := AMAX1(RAT,MAG/P);
      RELERR[JR] := B;
      ABSERR[JR] := EXP(POWER*LN(B));
    END;
  SHRUNK := FALSE;
  REPEAT
    WRITE('...');
    SHRUNK := FALSE;
    FOR J:=1 TO RN DO
      BEGIN
        IF ABSERR[J]<>0.0 THEN
          BEGIN
            P := 1.0;
            M := RN;
            FOR K:=1 TO RN DO
              BEGIN
                WRITE('...');
                IF K<>J THEN
                  BEGIN
                    DIST := CPABS(RWR[J]-RWR[K],RWI[J]-RWI[K]);
                    UNCERT := ABSERR[K];
                    CERT := DIST-UNCERT;
                    IF CERT>= ABSERR[J] THEN
                      BEGIN
                        P := P*CERT;
                        M := M-1;
                      END;
                  END;
              END;
            OLDERR := ABSERR[J];
            ABSERR[J] := RELERR[J]/P;
            IF M>1 THEN
              ABSERR[J] := EXP((1.0/M)*LN(ABSERR[J]));
            IF ABSERR[J]<OLDERR*0.99 THEN
              SHRUNK := TRUE;
          END;
      END;
  UNTIL SHRUNK=FALSE;
  SECHALF;
END; (* RBND2 *)

PROCEDURE RPQR(RNDEG:INTEGER; COEF:LISTRP1; VAR FIERR:INTEGER;
               VAR RWI,RWR:LISTR);

VAR J,K,RLOW,RIGH: INTEGER;
    RECIP: REAL;
    RM,RVEC: MATRIX;
    RS: STRING[80];
    CH: CHAR;
    RSCALE: LISTR;
```

```
      RINT: LISTI;

  BEGIN (* RPQR *)
    RIERR := 0;
    IF COEF[1]=0 THEN
      BEGIN
        RIERR := 2;
        WRITELN('LEADING COEFFICIENT IS ZERO IN RPQR');
        SPACEBAR;
        EXIT;
      END;
    IF COEF[1]=0.0 THEN
      RECIP := 1.0
    ELSE
      RECIP := 1.0/COEF[1];
    FOR K:=1 TO RNDEG DO
      BEGIN
        RA[1,K] := -COEF[K+1]*RECIP;
        FOR J:=2 TO RNDEG DO
          RA[J,K] := 0.0;
      END;
    FOR K:=2 TO RNDEG DO
      RA[K,K-1] := 1.0;
    BALANCE(RNDEG,RNDEG,RA,RLOW,RIGH,RSCALE);
    ELMHES(RNDEG,RNDEG,RLOW,RIGH,RA,RINT);
    ELTRAN(RNDEG,RNDEG,RLOW,RIGH,RA,RINT,RVEC);
    HQR2(RNDEG,RNDEG,RLOW,RIGH,RA,RWR,RWI,RVEC,RIERR);
    IF RIERR<>0 THEN
      BEGIN
        RIERR := 1;
        WRITELN('NO CONVERGENCE IN 40 QR ITERATIONS IN RPQR');
        SPACEBAR;
        EXIT;
      END;
  END; (* RPQR *)

BEGIN (* TTRPQR *)
  CLEARSCREEN;
  REPEAT
    CLEARIT(1);
    WRITE('WHAT IS THE DEGREE OF THE POLYNOMIAL? ');
    INTREAD(NDEG);
  UNTIL ((NDEG>0) AND (NDEG<=MAXDEG));
  PDEG := NDEG+1;
  FOR I:=1 TO MAXDEGP1 DO
    PRCOEF[I] := 0.0;
  CLEARIT(0);
  WRITELN('ENTER COEFFICIENTS OF POLYNOMIAL TERMS:      ');
  WRITE('                                ');
  GETPOLCOF(PDEG,PRCOEF);
  CLEARSCREEN;
  WRITELN('PLEASE WAIT ');
  RPQR(NDEG,PRCOEF,TIERR,TWI,TWR);
  CLEARSCREEN;
  IF TIERR=0 THEN
```

```
    BEGIN
      WRITELN;
      WRITELN('THE ARRAY OF COMPLEX ROOTS IS');
      WRITELN;
      FOR I:=1 TO NDEG DO
        BEGIN
          WRITE('[');
          WRITE(TWR[I]);
          WRITE(',');
          WRITE(TWI[I]);
          WRITELN(']');
        END;
      WRITELN;
      SPACEBAR;
      RBND2(NDEG,PRCOEF,TWR,TWI,TABSERR,TRELERR,TKLUST,TKEP);
      WRITELN;
      WRITELN('THE ARRAY OF ABSOLUTE ERRORS IS');
      WRITELN;
      FOR I:=1 TO NDEG DO
        BEGIN
          WRITE('[');
          WRITE(TABSERR[I]);
          WRITELN(']');
        END;
      WRITELN;
      SPACEBAR;
    END;
END; (* TTRPOR *)

MODEND.
```

```
(* VERSION 0207 *)
MODULE OVERLAY23;

EXTERNAL PROCEDURE GOTOXY(X,Y:INTEGER);

EXTERNAL PROCEDURE CLEARSCREEN;

EXTERNAL PROCEDURE SPACEBAR;


PROCEDURE HELP;

TYPE STRNG40 = STRING[40];

VAR INFO: ARRAY[1..12] OF STRNG40;
    I: INTEGER;

BEGIN (* HELP *)
  INFO[1] := 'TO DO MATRIX WORK ONE MUST FIRST KEY';
  INFO[2] := 'IN A MATRIX "A". WITH THIS ONE CAN';
  INFO[3] := 'CALCULATE ITS EIGENVECTORS, EIGEN-';
  INFO[4] := 'VALUES, INVERSE OR DETERMINANT.  ';
  INFO[6] := 'TO SOLVE A SYSTEM OF SIMULTANEOUS ';
  INFO[7] := 'EQUATIONS, "A" TIMES "X" = "B",';
  INFO[8] := 'ONE MUST ALSO KEY IN A MATRIX "B".';
  INFO[5] := ' ';
  INFO[9] := ' ';
  INFO[10] := 'THIS MATRIX "B" MAY BE KEYED IN AFTER ';
  INFO[11] := 'THE MATRIX "A" HAS BEEN KEYED IN AND ';
  INFO[12] := 'WORKED WITH.  ';
  CLEARSCREEN;
  GOTOXY(0,0);
  WRITE('DIRECTIONS FOR INPUT: ');
  GOTOXY(0,3);
  FOR I:=1 TO 12 DO
    WRITELN(INFO[I]);
  GOTOXY(0,20);
  SPACEBAR;
END; (* HELP *)

MODEND.
```

```
(* VERSION 0286 *)

MODULE OVERLAY24;

(* EIGENVEC MODULE *)

TYPE DOMAIN1 = 1..20;
     MATRIX = ARRAY[DOMAIN1,DOMAIN1] OF REAL;
     LISTI = ARRAY[DOMAIN1] OF INTEGER;
     LISTR = ARRAY[DOMAIN1] OF REAL;


VAR IORDIM: EXTERNAL INTEGER;
    ICTA,TVEC: EXTERNAL MATRIX;
    TEVR,TEVI: EXTERNAL LISTR;


    EXTERNAL PROCEDURE CLEARSCREEN;

    EXTERNAL PROCEDURE SPACEBAR;

    EXTERNAL FUNCTION YES: BOOLEAN;

    EXTERNAL [5] PROCEDURE HQR2(NM,N,LOW,IGH:INTEGER;VAR H:MATRIX;V
                                ,WI:LISTR;
                                VAR Z:MATRIX;VAR IERR:INTEGER);

    EXTERNAL [6] PROCEDURE BALANCE(NM,N:INTEGER;VAR A:MATRIX;VAR LO
                                   :INTEGER;
                                   VAR SCALE:LISTR);

    EXTERNAL [6] PROCEDURE ELMHES(NM,N,LOW,IGH:INTEGER;VAR A:MATRIX;
                                  INT:LISTI);

    EXTERNAL [6] PROCEDURE ELTRAN(NM,N,LOW,IGH:INTEGER;VAR A:MATRIX;
                                  LISTI;
                                  VAR Z:MATRIX);

    EXTERNAL [6] PROCEDURE BALBAK(NM,N,LOW,IGH:INTEGER;SCALE:LISTR;M
                                  INTEGER;
                                  VAR Z:MATRIX);

    EXTERNAL [3] PROCEDURE HARDCOPY(HNUM:INTEGER);

PROCEDURE TTRNAA;

VAR QUITFLAG: BOOLEAN;
    I,J,PDEG,NDIM: INTEGER;
    CHOICE,SELECTION: CHAR;
    TCOEF: LISTR;
    TWI,TWR: LISTR;
    TIERR,NDEG: INTEGER;
    CH: CHAR;
    SREEL: STRING;
    P4: INTEGER;
```

```
  TA: MATRIX;


PROCEDURE RNAA(RNDIM,RN:INTEGER;VAR RA:MATRIX;VAR REVR,REVI:LISTF;
               VAR RVEC:MATRIX;VAR RIERR:INTEGER);

VAR SS: STRING;
    RSCALE: LISTR;
    INT: LISTI;
    I,J,RLOW,RIGH: INTEGER;
    CH: CHAR;


  PROCEDURE ERRMSS2;
  BEGIN (* ERRMSS2 *)
    WRITELN('MORE THAN 100 QR ITERATIONS NEEDED ');
    WRITELN('FOR SOME EIGENVALUE IN RNAA.');
    SPACEBAR;
    EXIT;
  END; (* ERRMSS2 *)

BEGIN (* RNAA *)
  BALANCE(RNDIM,RN,RA,RLOW,RIGH,RSCALE);
  ELMHES(RNDIM,RN,RLOW,RIGH,RA,RINT);
  ELTRAN(RNDIM,RN,RLOW,RIGH,RA,RINT,RVEC);
  HQR2(RNDIM,RN,RLOW,RIGH,RA,REVR,REVI,RVEC,RIERR);
  IF (RIERR<>0) THEN
    ERRMSS2;
  BALBAK(RNDIM,RN,RLOW,RIGH,RSCALE,RN,RVEC);
END; (* RNAA *)

PROCEDURE PRVECTORS;
VAR I: INTEGER;

BEGIN (* PRVECTORS *)
  J := 1;
  REPEAT
    IF TEVI[J]<>0.0 THEN
      BEGIN
        WRITELN('VECTOR ',J,' HAS COMPONENTS ');
        FOR I:=1 TO NDIM DO
          BEGIN
            WRITE('VECTOR',J,'[',I,']=','[',TVEC[I,J]);
            WRITELN(',',TVEC[I,J+1],']');
          END;
        WRITELN;
        SPACEBAR;
        WRITELN;
        WRITELN('VECTOR ',J+1,' HAS COMPONENTS ');
        FOR I:=1 TO NDIM DO
          BEGIN
            WRITE('VECTOR',J+1,'[',I,']=','[',TVEC[I,J]);
            WRITELN(',',-TVEC[I,J+1],']');
          END;
```

```
            WRITELN;
            J := J+2;
          END
        ELSE
          BEGIN
            WRITELN('VECTOR ',J,' HAS COMPONENTS ');
            FOR I:=1 TO NDIM DO
              BEGIN
                WRITE('VECTOR',J,'[',I,']=','[',TVEC[I,J]);
                WRITELN(',0.000000000000000E+000]');
              END;
            J := J+1;
          END;
        SPACEBAR;
        WRITELN;
      UNTIL J>NDIM;
    END; (* PRVECTORS *)


  BEGIN(* TTRHAA *)
    CLEARSCREEN;
    NDIM := ICNDIM;
    FOR I:=1 TO NDIM DO
      FOR J:=1 TO NDIM DO
        TA[I,J] := ICTA[I,J];
    CLEARSCREEN;
    WRITE('PLEASE WAIT...');
    RHAA(NDIM,NDIM,TA,TEVR,TEVI,TVEC,TIERR);
    CLEARSCREEN;
    WRITELN('THE ARRAY OF COMPLEX EIGENVALUES IS');
    WRITELN;
    FOR I:=1 TO NDIM DO
      BEGIN
        WRITE('[');
        WRITE(TEVR[I]);
        WRITE(',');
        WRITE(TEVI[I]);
        WRITELN(']');
      END;
    WRITELN;
    SPACEBAR;
    WRITELN;
    WRITELN('THE COMPLEX EIGENVECTORS ARE ');
    WRITELN;
    PRVECTORS;
    WRITELN;
    WRITE('DO YOU WANT A HARD COPY? Y/N ');
    IF YES THEN
      HARDCOPY(1);
  END;(* TTRHAA *)

  {$DEND.
```

DISTRIBUTION LIST

PROFESSOR HAROLD FREDRICKSEN
Code 53Fs
DEPARTMENT OF MATHEMATICS
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

PROFESSOR MICHAEL HARTMANN
Code 53Hw
DEPARTMENT OF MATHEMATICS
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

PROFESSOR GORDON E. LATTA
Code 53Lz
DEPARTMENT OF MATHEMATICS
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

PROFESSOR RAUL MENDEZ
Code 53Mu
DEPARTMENT OF MATHEMATICS
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

PROFESSOR BERT RUSSAK
Code 53Ru
DEPARTMENT OF MATHEMATICS
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

PROFESSOR MAURICE D. WEIR
Code 53Wc
DEPARTMENT OF MATHEMATICS
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

PROFESSOR CARROLL WILDE
Code 53Wm
DEPARTMENT OF MATHEMATICS
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

PROFESSOR LARRY WILLIAMSON
Code 53Wj
DEPARTMENT OF MATHEMATICS
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

DEPARTMENT OF MATHEMATICS
Code 53
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

DEFENSE TECHNICAL INFORMATION (2)
CENTER
CAMERON STATION
ALEXANDRIA, VIRGINIA 22214

LIBRARY, Code 0142 (2)
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

RESEARCH ADMINISTRATION OFFICE
Code 012
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943